AFRL-RI-RS-TR-2013-038

MIXED-INITIATIVE COA CRITIC ADVISORS (MICCA)

RAYTHEON/BBN TECHNOLOGIES CORP.

FEBRUARY 2013

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2013-038 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

DALE W. RICHARDS
Work Unit Manager

/ S /

JULIE BRICHACEK, Chief
Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
FEB 2013	FINAL TECHNICAL REPORT	JUN 2010 – SEP 2012

4. TITLE AND SUBTITLE

MIXED-INITIATIVE COA CRITIC ADVISORS (MICCA)

5a. CONTRACT NUMBER
FA8750-10-C-0184

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Alice M. Mulvehill, Fusan Yaman, Brett Benyo

5d. PROJECT NUMBER
S2MC

5e. TASK NUMBER
MI

5f. WORK UNIT NUMBER
CA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Raytheon/BBN Technologies Corp
10 Moulton Street
Cambridge, MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISC
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2013-038

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2013-0503
Date Cleared: 5 Feb 2013

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Mixed Initiative courses of action (COA) Critic Advisor (MICCA) research effort developed an agent-based proof of concept prototype for evaluation and alignment of externally generated or historical plans and/or COAs with the current world state. MICCA was used to support the evaluation and adaptation of plans from two domains, the Rovers domain from the Third International Planning Competition and the DARPA Joint Air/Ground Operations Unified Adaptive Replanning (JAGUAR) Air Tasking Order domain that supports dynamic tactical air mission planning and execution. MICCA software agents operate in a domain independent manner, and communicate via a blackboard framework. Domain dependent information is utilized to support some of their reasoning. Additional work included the development of translation tools to support the interpretation of domain specific input, the development of a variety of user interface capabilities to facilitate mixed-initiative decision making, and the combination of generative and case based planning approaches to support both evaluation and adaptation of plans.

15. SUBJECT TERMS
Agent Based Systems, Mixed-Initiative Planning, Couse of Action (COA) Evaluation, Planning, Case Based Reasoning, Blackboard Systems, Hierarchical Task Networks, Quantitative Temporal Reasoning, Preference Model

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**DALE W. RICHARDS**
U	U	U	SAR	73	19b. TELEPONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

SECTION **PAGE**

LIST OF FIGURES

LIST OF TABLES

TABLE **PAGE**

1.0 SUMMARY

This final report describes the Mixed Initiative Course of Action (COA) Critic Advisor (MICCA) research effort. MICCA is an agent-based proof of concept prototype that has been developed to support the evaluation and alignment of externally generated or historical plans and/or COAs with the current or some projected world state. To date MICCA has been used to support the evaluation and adaptation of plans from two domains, the Rovers domain that was one of the domains used during the Third International Planning Competition and the DARPA Joint Air/Ground Operations Unified Adaptive Replanning (JAGUAR) Air Tasking Order (ATO) domain that supports dynamic tactical air mission planning and execution. The MICCA agents operate in a domain independent framework, communicate through a blackboard, and utilize domain dependent information to support some of their reasoning. Additional aspects of this work include the development of translation tools to support the interpretation of domain specific input, the development of a variety of user interface capabilities to facilitate mixed-initiative decision making, and the combination of generative and case based planning approaches to support both evaluation and adaptation. This report provides a technical overview of MICCA, a synopsis of the related research, a description of how MICCA was used in each domain, some of the metrics used to evaluate MICCA's performance, and recommendations for future research and development.

2.0 INTRODUCTION

MICCA is a proof of concept agent-based software prototype that was developed to aid human operators in evaluating/critiquing, adapting and aligning past military plans to meet current objectives and constraints. MICCA operates in a publication/subscription environment. MICCA uses the Distributed Episodic Exploratory Planning (DEEP) publication/subscription blackboard [1] that was developed by the Air Force Research Laboratory (AFRL). To date, MICCA has been used to support the evaluation and adaptation of plans from two domains, the Rovers domain (a benchmark domain introduced in the Third International Planning Competition) and the DARPA Joint Air/Ground Operations Unified Adaptive Replanning (JAGUAR) Air Tasking Order (ATO) domain [2].

At an abstract level, MICCA enables commanders to readily access and leverage historical data from distributed sources for use in their decision making, e.g., military mission planning. Specifically, MICCA supports the evaluation of one or more courses of action (COAs) using domain independent and user-defined evaluation criterion and adaptation methods. A COA describes a set of problem solving steps that can be used to achieve an objective or solve a specific problem. In MICCA, a COA is syntactically equivalent to a hierarchical plan. In this report, the terms COA and plan will often be used interchangeably.

Fully automated systems operate best when the domain is fully specified and/or when the problem is under-constrained. When a fully automated system makes a wrong choice and reaches a dead end, its software will trigger a backtracking cycle. With standard backtracking, the system will visit previous states in order and try every possible option before reevaluating the decisions made earlier. More sophisticated backtracking algorithms try to identify the most likely steps that led to the dead end and skip the exploration of options in between the final and identified steps. These algorithms, in general, only partially prune the search space and require a lot of book keeping.

MICCA is a mixed initiative system. Mixed-initiative systems involve humans who can often easily spot the source of the problem and suggest modifications, which can readily direct the system back onto the right track. In many mission planning contexts, the human planners have access to a variety of information. Some of this information is available from services such as databases, but some of the information is not available to an automated system (such as experiences or hunches). While there have been a number of efforts aimed at automating decision making, the process is costly and does not guarantee that all of the knowledge is captured. In a mixed-initiative system, the human user participates in the problem solving which can generally reduce the size of the search space to a tractable problem. To aid the user in understanding how MICCA has generated results, and thus gain confidence in the decisions and evaluations MICCA makes, MICCA provides an explanation for every choice, score, and adaptation performed. A variety of tools are available in MICCA to enable the human operator to influence the behavior of the system through expressed preferences.

Since MICCA is intended to operate in a larger command and control environment and to operate on a variety of planning situations and time frames, e.g., tactical, operational and strategic, the MICCA agent framework has been designed to be domain independent. MICCA agent capabilities can be extended with domain specific information, either through access to

domain specific knowledge bases and/or through mixed-initiative interaction with a user. In this report, we describe the current MICCA capabilities.

2.1 Background

This section provides the background knowledge on technologies and formalisms used in MICCA.

2.1.1 HTN planning

There are many technical approaches that can be utilized to support the development of a plan. For each approach, there needs to be some method to allow software or a human to assess the current world state, to specify a goal or objective, and to specify and link actions in a sequence, each with time and resource specifications suited to achieving the goals.

Generative planning approaches, like SHOP and JSHOP can support the generation of a plan from scratch. SHOP and the Java based version, JSHOP [3] is a hierarchical task planner that utilizes the Hierarchical Task Network (HTN) and models about the domain to generate plans. A plan can also be derived from one or more historical plans. Experience-based planning techniques such as case based reasoning (CBR) can be used to find similar historical plans and adapt them to operate in a current context. Hybrid approaches that combine generative methods with case based methods can also be used to generate a plan [4].

HTN planning couples well with case-based planning and works well within mixed-initiative planning frameworks because task methods align well with cases, and are easily understood by people. The HTN approach is also suitable for plan revisions. For MICCA, we have adopted a hybrid approach. Because the focus of MICCA is not on the generation of the plan, the MICCA process starts with a set of historical plans that are stored in a CBR system. Similarity matching is used to allow MICCA agents to find historical plans that match part or all of a given objective specification. The case base is also used to support some of the alignment of the historical plan with the current context.

The HTN technology is used to support the adaptation of candidate historical plans with the world state. In our work, we utilize domain-specific HTN models. An HTN planning domain is a list of operators (simple actions that can be directly executed by an agent) and methods (decomposition rules guarded by applicability conditions).

Figure 1 contains an example that demonstrates the methods for a simple travel domain. In this figure, the tasks in oval are high level and the tasks in rectangles are simple (i.e., corresponding to an operator). In this example, the taxi travel is decomposed into three simple tasks that can be achieved by operators whereas the air travel is decomposed into a mixture of simple and high-level tasks. An HTN planning problem is defined in terms of tasks to be achieved and a plan is generated by recursively decomposing the tasks using the methods until every task corresponds to an operator. The methods are essential for the planning algorithm because if there is no method for a high level goal/task then no plan can be generated.

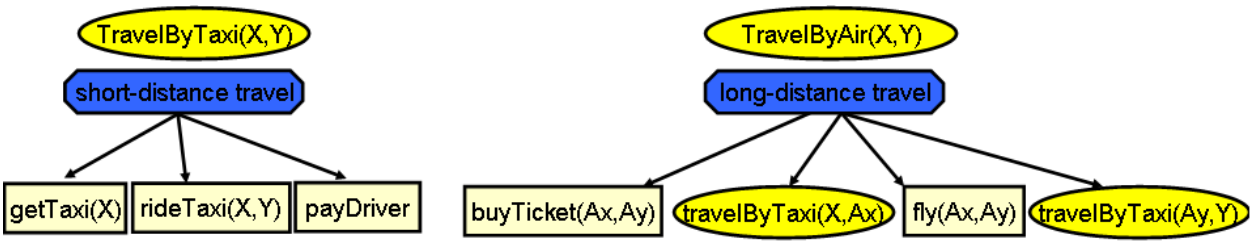

Figure 1. Visualization of two JSHOP methods for a simple travel domain where the two high level tasks are traveling by taxi and traveling by air.

Ordinarily a domain expert provides the methods, which is a costly and time-consuming process. For this project we have leveraged existing HTN domains (the freely available Rovers domain developed by the University of Maryland during the Third International Planning Competition) and some of the hierarchical process models that were created in the DARPA JAGUAR program and converted to HTN methods and operators to operate in MICCA.

2.1.2 Quantitative Temporal Reasoning

In real world applications conditions change rapidly and punctuality is not often possible. For this reason we need to maintain temporally flexible plans in which the start and end times of each action, as well as their duration, are intervals instead of fixed values. Such plans have proven to be useful in NASA's deep space missions [5] and in the development of personal planning agents [6, 7].

Simple Temporal Networks (STNs) [8] are a tractable subclass of quantitative temporal networks. An STN is a directed graph where vertices are time points and every arc represents a single temporal constraint defined on two time points.

Figure 2 shows the STN for the constraints of a ship that needs to leave port A sometime during the (4, 5) interval and arrive at port B sometime in (15, 30). The ship takes (10, 20) units of time to sail from A to B. In this figure t0 represents the beginning of time, t1 and t2 denote the departure times from A and the arrival time at B respectively.

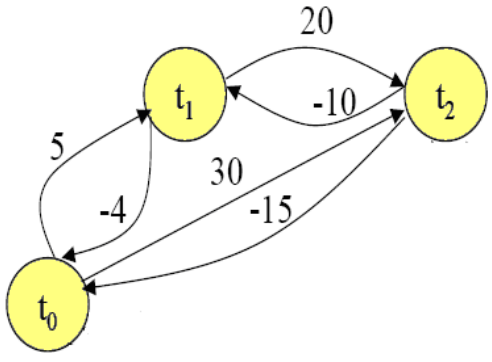

Figure 2. STN for representing the temporal constraint of a ship's sail plan.

Quantitative temporal constraints allow users to express numeric constraints such as "event A should end at least 15 minutes before event B starts". This is more powerful than qualitative constraints such as Allen constraints [9] where one can only say A should end before B starts.

2.1.3 Preferences

Preference models are a method that can be employed to rank COA options based on evaluation scores along different dimensions. In MICCA, lexicographic preference models (LPMs) are used. An LPM defines an order of importance on the measure variables (criteria) and uses it to make preference decisions. For example when picking an airbase for an emergency landing, the most important criterion for a military mission is to pick an airbase that is located in a friendly country over an enemy one. The second most important criterion is to pick an airbase that can support the landing and possible ground maintenance of the aircraft.

LPMs are simple yet intuitive preference representations; it has been shown that humans indeed employ LPMs in decision making [10, 11]. In addition, the computational complexity of reasoning with LPMs is very low. For example comparing two choices with respect to an LPM is linear in the number of criteria. The main drawback of this approach is the inability to represent conditional preferences.

Historically LPMs have been used on numeric, Boolean, or discrete valued attributes. In our research we have utilized an extended version of LPMs to handle attributes within a monotonic continuous domain (such as the scores generated by evaluation agents). Yaman and desJardin [12] have previously demonstrated how such extensions can be achieved for another preference model representation. The technique is directly applicable to LPMs and does not add any complexity in reasoning.

2.1.4 Case Based Planning

Reuse of historical plans is a common strategy that is employed to solve problems where time is limited, there is uncertainty about the current and/or future state of the world, or the human decision maker lacks sufficient domain expertise to solve the current problem. While the retrieval of similar historical plans can be facilitated by sophisticated search engines, evaluating the usefulness of historical plans is more complicated, especially when the historical plan is old and/or when it was created for a different context and/or by a different user. Additionally, evaluation of usefulness can be very subjective and context sensitive.

Case based planning systems rely on a history of past experiences to support problem solving. The underlying technology is called case based reasoning (CBR) and the reasoning cycle [13] is comprised of the following functions: retrieve, reuse, revise, and retain. The past experiences are stored in specialized repository called a case base. A case is a single experience or episode with a set of features. Features are attribute-value pairs that describe the main characteristics of the case, e.g., type, time of occurrence, actors involved, name of activities, name of resources, etc. Features with a higher degree of "importance" typically define the essential or distinguishing features of the case and can act as weights to bias retrieval. An example is the "limiting factors" in a COA that might describe some environmental feature that inhibits the execution of one or more tasks associated with the COA. Since features describe the case, the most common usage of the features in a CBR system is to support similarity matching and case retrieval.

The main benefit of CBR systems is to allow a user to apply previous experience to current problem solving contexts. Because the past and the present are rarely equivalent, the previous experience/case must be revised to fit the needs of the current problem. Revision can be as simple as changing the temporal and spatial values of a past case or the required revisions may be more complicated. For example, often there is no single match but instead parts of several previous experience/cases match the needs of the current problem solving context. In these situations, revision may involve interleaving or merging together parts of several retrieved cases to form a single usable case.

MICCA utilizes CBR technology to support its evaluation and instantiation activities. In MICCA the case base is used as a historical plan repository to create the initial set of candidate plans/COAs. It is also used by the MICCA instantiation agents to support the alignment of historical plans with the current world state and to support the merging of multiple retrieved candidates that cover only part of the objective into a single usable plan that can satisfy all of the current objectives.

2.1.5 DEEP

The MICCA agents were developed to operate within a larger decision support environment called Distributed Episodic Exploratory Planning (DEEP). DEEP originated as an Air Force Research Laboratory (AFRL) program. The vision behind DEEP is to provide a mixed initiative decision support environment where commanders can readily access and leverage historical data from distributed sources for use in decision making. The DEEP vision includes a construct of keeping plans "alive" by using a set of plan critics that have access to the current world state and can evaluate what needs to change in the plan as the world state changes. All DEEP agents communicate through a blackboard. Figure 3 displays the DEEP vision and the following is an excerpt from Carrozoni [1] that explains the DEEP vision:

"*The starting point for entry into the system is the commander using a planning agent. The planning agent allows for the commander to input information into the system which defines their current objectives. These objectives, along with other information, such as resources, locations, and time constraints, are collectively known as the situation. This situation is then placed on the blackboard. The blackboard then notifies all registered components of the new situation. The other planning agents, with their associated case bases (3) and cased-based reasoning, search their case base using the situation given for relative past experiences.*

These results are then modified to fit the current situation (4) and are posted to the blackboard (5). Once the "candidate plans" are on the blackboard, they are adapted by specialized agents to further refine these plans (6). These plans are now ready for critique by the critic agents. These agents concurrently scrutinize the plans and score them based on their individual expertise (7). Once the plans are scored, the execution selection critic gathers the adapted plans along with their scores, determines their overall scores, and selects a number of top rated plans to be executed (8). The top rated plans are now run against a simulation (9). Now that there is a new plan which has been simulated, the results of this simulation are assimilated with the plan and stored in the case base. The process is not over at this time, but instead the architecture can allow the plans to be run through the cycle many more times if desired (10)."

As visualized in Figure 3 [1], within this vision MICCA provides the Evaluation, Adaptation and Execution agents as well as the meta-critic framework (highlighted with the Blue background in Figure 3). It is important to note that MICCA is a stand-alone system and independent from DEEP. MICCA is agnostic to the source of the input, i.e., the candidate plans can be historical or produced by an external tool. The current MICCA implementation uses the DEEP blackboard, but this is not a requirement for MICCA functionality. A different blackboard or messaging architecture could be used instead.

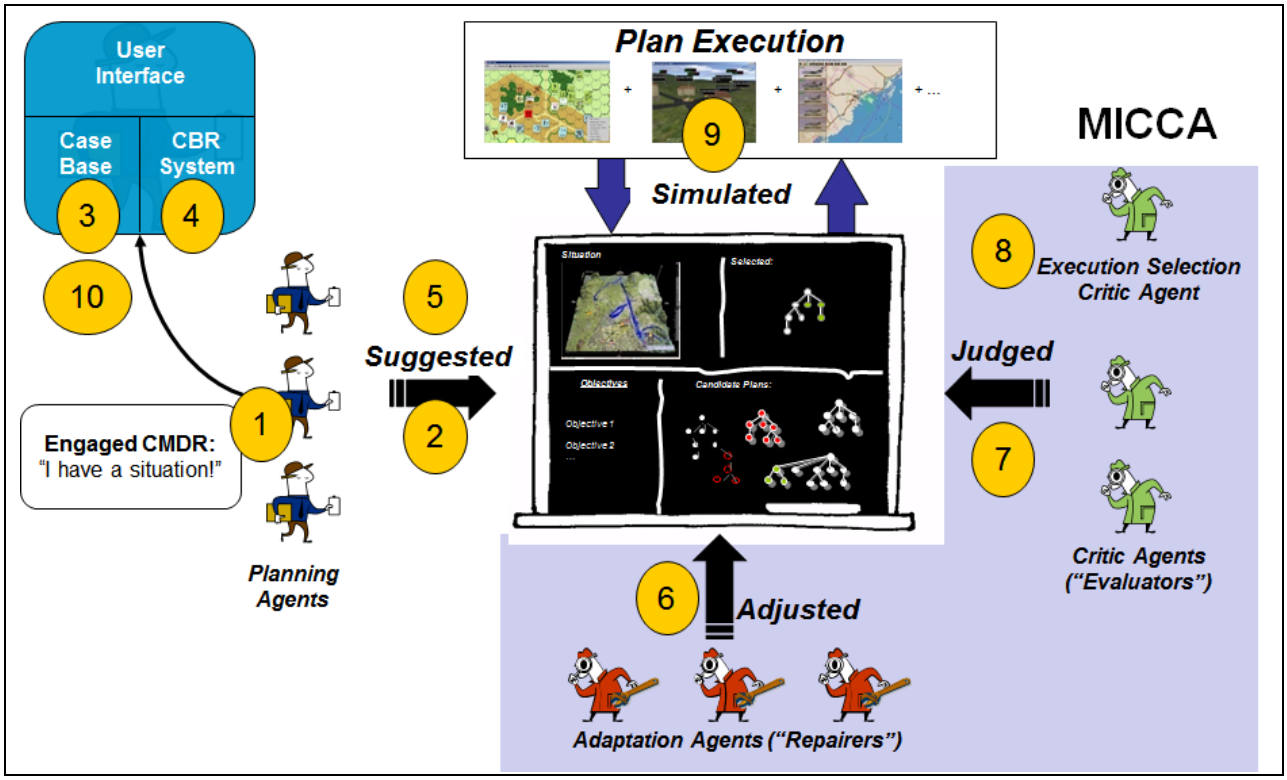

Figure 3. DEEP vision.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

MICCA is an agent-based system that supports the evaluation and adaptation of a set of plans that can be used to generate a course of action that satisfies a current set of objectives. MICCA can evaluate candidate plans that have been developed by an external system or historical plans that are retrieved from a case repository where access to the tool that created the historical plans is not provided.

The MICCA prototype system is comprised of several different types of agents that interact within an agent framework to process candidate plans/COAs. MICCA agents subscribe to the DEEP blackboard to obtain planning products, such as the world state and objectives. MICCA agents can also publish results back to the DEEP blackboard for use by some external COA processing or planning system. The DEEP blackboard is also used to support MICCA agent communication. The agents communicate through asynchronous message passing.

While many of the agents automatically process the plans, several custom Graphical User Interfaces (GUIs) are available to allow the user to view results, to create agents, to influence how the agents act (policies), to generate reports, and to publish selected COAs for use by a given external planning and execution system. Section 3.6 contains a table of functions and references to various figures throughout this paper that showcase the user interfaces that have been developed during this research.

The MICCA process starts with the receipt of a problem, which is comprised of a set of goals or objectives, a textual statement about the commander's intent (CI), and world state information. The objectives consist of a set of goals that need to be satisfied. Each objective can be mapped back to some statement in the CI. The candidate COAs are provided from a historical repository which is implemented as a case base. Each COA in the case base is comprised of a set of attribute value pairs that constitute the descriptive features of the case along with plan execution data. A specialized enabling agent called the *Case Base Retrieval* agent uses the objective information to form queries against the historical repository and retrieve relevant candidate COAs. MICCA agents utilize information about the world state and relevant domain information to evaluate and adapt the candidate COAs.

Figure 4 presents a schematic that describes how the evaluation cycle (previously called "critique") is performed in MICCA. Case Base agents use the goal/objective information to search the historical case base for similar plans that are published to the blackboard (BB). Upon receiving the candidate plans through the DEEP BB, MICCA evaluation agents score each candidate plan in terms of general criteria such as cost, adaptability, and risk. The evaluation agents can operate in parallel and they do not communicate or depend on each other for operation. The comparison agent compares the plans pairwise using the scores from the evaluation agents, as well as the operator-specified preferences that are specified in a Lexicographic Preference Model (LPM). The evaluated COAs are ranked and displayed to the user by the Ranking Agent. A number of GUIs have been created to allow the user to view and modify the LPM, to create or modify evaluation agents, set the preference models and display specific reports.

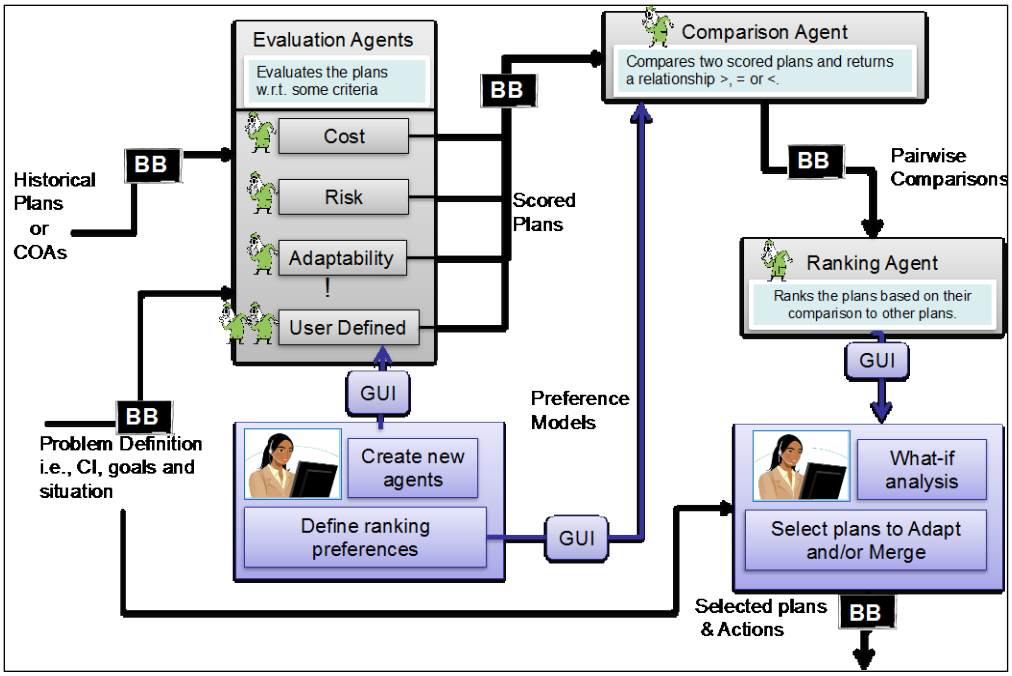

Figure 4. Mixed initiative evaluation cycle in MICCA: Each COA is evaluated from different dimensions and the COAs are ranked with regard to the given preference model. The user makes the final selection of the COAs.

The next step is to select a subset of the COAs for alignment with the current objective constraints that are defined in the world state or domain models thus improving their quality. The COA selection can be done automatically by MICCA or interactively with the human operator. The alignment process is called "*adaptation*" in MICCA. The *Instantiation Agents* serve as a preprocessing step for the adaptation cycle (See Figure 5).

The function of the instantiation agents is to map parts of the candidate objective(s) to the current objective(s) and edit the candidate plan to reflect that mapping. Instantiation agents also replace old or unavailable resources with current resources. The *Merging* agent provides a technique for merging elements from multiple candidate plans together to produce a plan that covers more of the current objectives.

After the instantiation and merging steps, the coordination agent ensures that the COAs are passed to the next adaptation agent in the order defined by the adaptation policy, which can be edited by the user. Currently two MICCA adaptation agents, called Plan and Temporal, can be used to resolve causal relationships and temporal constraints. If MICCA encounters a conflict, i.e., the set constraints after the adaptations are unsolvable, the user may be requested to participate in the de-confliction process through a custom designed GUI (See Figure 16). All adapted COAs are then re-evaluated and displayed in a ranked list and presented to the user (See Figure 8). Here the user can request plan details and justification regarding MICCA agent actions. Through the Ranking Agent GUI, the user can select one or more Revised COAs (RCOAs) to publish (to DEEP) for continued planning by some external subscribing planning system.

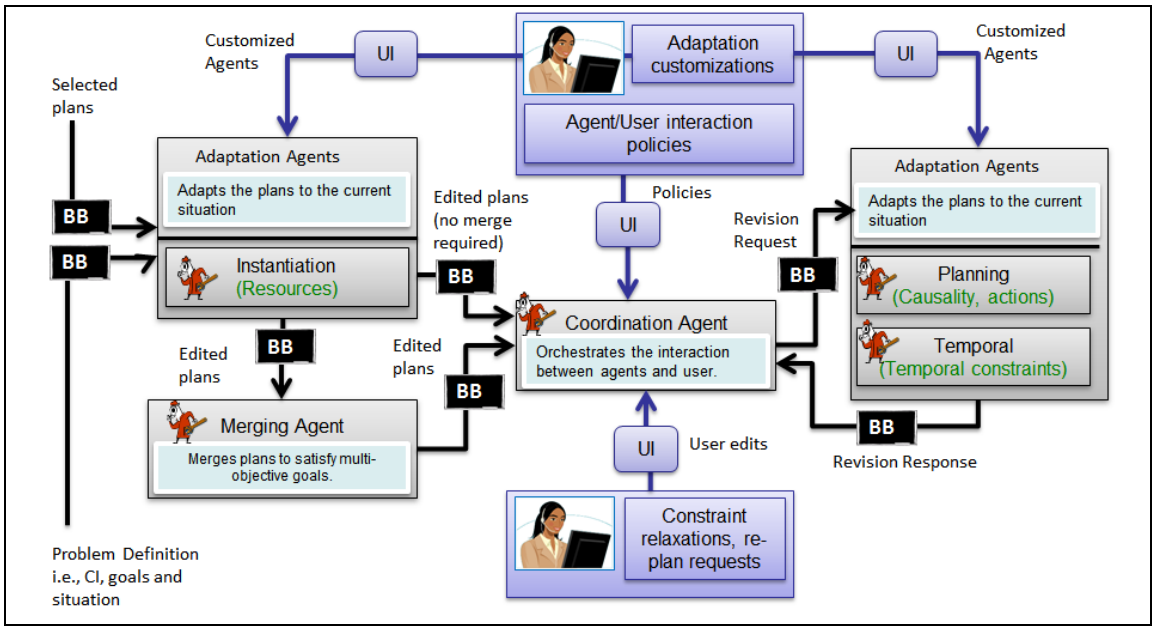

Figure 5. Mixed initiative adaptation cycle in MICCA: Several adaptation agents work on the selected COAs to fix the resource, causal and temporal constraints to make the COAs more viable. The user is involved in resolving de-conflictions.

The following subsections provide more detail about the various MICCA agents and the input/output requirements of each agent.

3.1 Plan Evaluation Agents

There are two basic inputs for the evaluation agents; a set of one or more plans as retrieved from the case base, and a set of one or more plans that are revised and published by the adaptation agents for subsequent evaluation. Table 1 contains a list of several evaluation agent classes and summarizes the scoring criteria they use to evaluate the candidate COA/plan or the revised COA/plan.

Table 1. Evaluation agents and their scoring criteria based on the plan/COA type.

Agent	Candidate COA/Plan	Revised COA/Plan
Cost	Sum of costs of each action in COA	Sum of costs of each action in COA
Adaptability (*domain specific*)	Number of action preconditions that are satisfied in the current state	Number of plan fragments that are reused
Risk	Number of anomalies encountered/expected and number of risky actions	Number of anomalies encountered/expected and number of risky actions.
Executability	N/A	Minimum temporal execution window
Completeness	Number of objectives covered	Number of objectives covered

The distinction between the retrieved candidates and revised plans is that the *retrieved COAs/plans* are historic plans that are obtained as a result of a query against a case base, whereas the *revised COAs/plans* are historic plans that are specifically tailored to satisfy the current

objective(s) within the current state. While evaluating the retrieved plans the goal is to assess how useful these historic plans are as starting points for the development of plans that will achieve the current objective. The evaluation of revised plans focuses on the quality of the plan as it is adapted and/or revised to achieve the current objective. The type of evaluation performed by the agent can be different for revised and retrieved plans. Similarly the set of agents used for evaluating the retrieved plans might be different than the agents used for evaluating the revised plans.

All evaluation agents, except the Adaptability agent, are domain independent. The Adaptability agent has been implemented to operate in each of the two different domains used to date – Rovers and JAGUAR. Domain specific knowledge is used to determine what it means to be similar within the given domain and this is important in deciding whether a historic plan is useful as the basis for satisfying a current objective. A customizable version of the Adaptability agent can be implemented in MICCA through tools that let the user define parametric queries which can be passed as an input to the agent. Alternatively, given the HTN models of a domain the agent might utilize the method and action preconditions to determine similarities.

The architecture also supports the development of specialized instances of a template based evaluation agent. This agent computes its score using the weighted sum of user selected plan features which are computed during the adaptation cycle or part of the possible meta-data the plans might have. This custom evaluation agent generation capability in MICCA, as displayed in the right pane of the display in Figure 6, gives the user flexibility to define new agents as needed or to use ones that have been previously defined.

Note that additional inputs might be necessary for specific evaluation agents. These external sources can take the form of a database, a simple file, or an information service. For example a more sophisticated cost agent may need to access a database for querying the material cost of the resources utilized in the plan. Or, if the risk agent needs weather information it may query an information service like a weather forecast service to determine the current or projected weather conditions for the evaluated plans. Knowledge bases are another source of input to the Evaluation agents. These are described in more detail in Section 3.8 and Appendix B.

The output of the evaluation agents is uniform for both revised and retrieved plans. The execution agent will take into account the value of the source of the plan to identify if the score is for a historic retrieved plan or for a revised plan as this can influence choices for employing the right preference model.

3.2 Execution Agents

As shown in Figure 4, the *Comparison Agent* and the *Ranking Agent* are examples of execution agents that process the evaluated candidate plans. The inputs to these agents include: a set of scored plans and a Lexicographic Preference Model (LPM). The output is a ranked list of plans which is presented to the user by the Ranking Agent (see Figure 8 for an example). The user then selects what plans he/she wants to publish for adaptation.

3.2.1 Comparison Agent

This agent compares a plan to other plans with the same context when all the scores of the plan are available. The comparison agent organizes the scores with respect to (w.r.t.) the context and the plan. The context of the plan also identifies the preference model that will be used for comparing plans within the same context. For example, a different LPM might be used for evaluating plans from different domains. Similarly the comparison of revised and retrieved plans might also need different preference models.

The ranking is based on the scores from one or more evaluation agents along different dimensions as defined in the LPMs. Figure 6 displays how the LPM can be defined and/or changed by the user. Note: this interface can also be used by the user to create new evaluation agents which can then be used to influence the preference model.

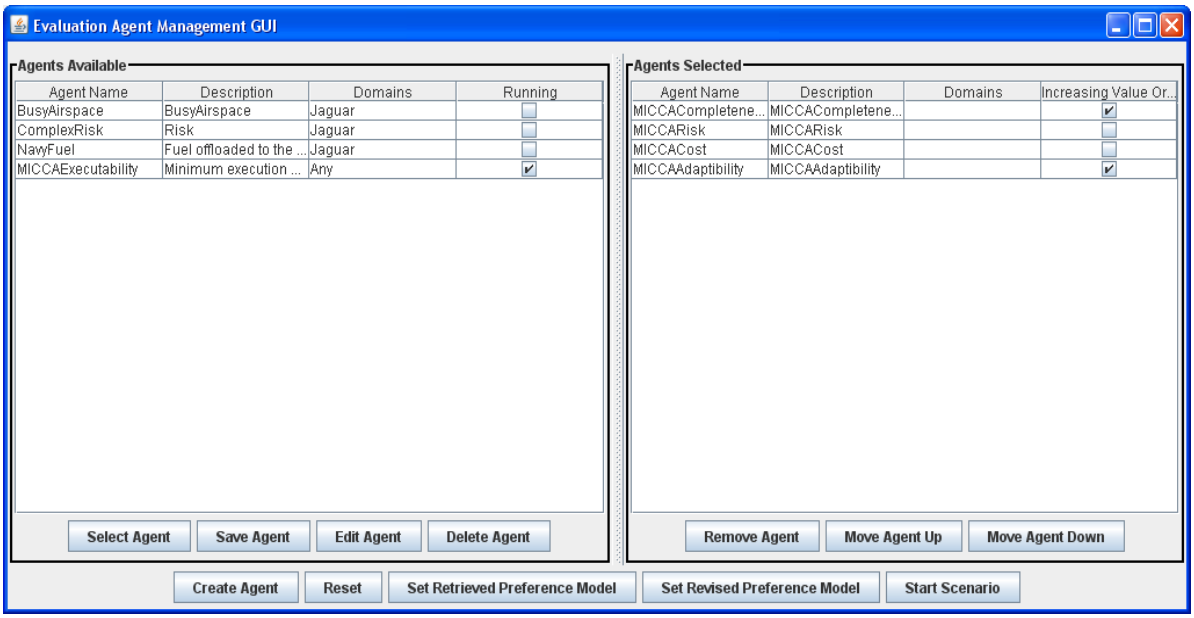

Figure 6. Setting the LPM and creating Agents.

Typical steps performed by the Comparison agent include:

- Once the Evaluation agents post Scored Plan objects on the blackboard, the Comparison agent processes each scored plan to maintain all scores related to a plan.
- Once **all** scores are in for a plan, that plan can be compared to other **related** plans.
  - The preference model that is input to the agent will identify the relevant scores thus the agent will know which scores are essential for the comparison and will be able to determine if **all** the scores are in or not.
  - The **related** plans are the ones that share the same Source, Objective, and State. It is meaningful to compare a plan with only the plans sharing the same context and these three parameters uniquely define a context for the plan.
- The Comparison agent posts pair wise comparison results for plans which are then posted on the BB as PlanPairComparison objects.

3.2.2 Ranking Agent

The Ranking agent processes the output of the Comparison agent. This agent is completely domain independent. It will compute a total order on the plans that is consistent with the pair wise ordering. This order (ranking) is displayed to the user in the Ranking Agent GUI. The ranking agent allows for several types of user interactions. The user can pick one or more plans from the ranking list to be further processed, i.e., revised or executed. The user can control what information is displayed about the plans through the interface by adding or removing data columns. Figure 7 is an example of the Ranking Agent GUI for the Rovers domain while Figure 8 is an example of the Ranking Agent GUI for the JAGUAR domain.

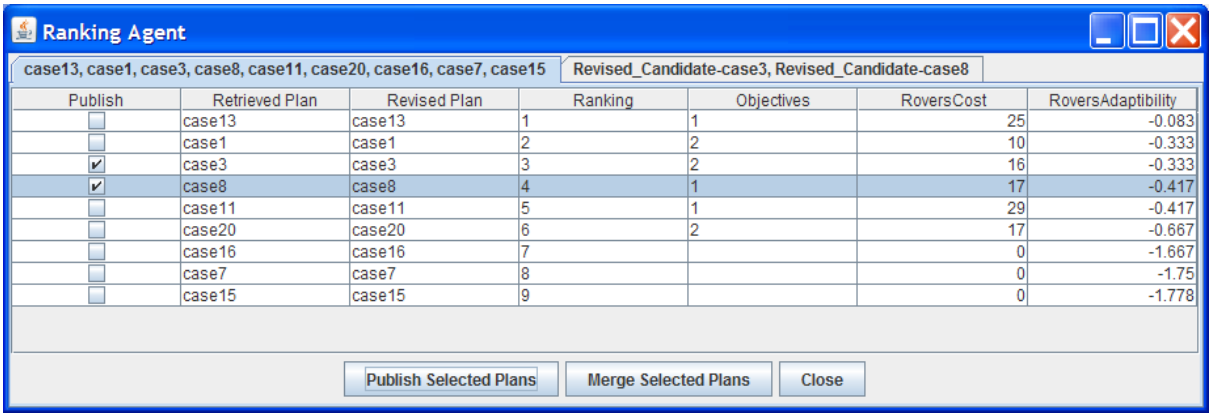

Figure 7. Ranking Agent displays the Retrieved and Revised Candidate Plan data for the Rovers domain.

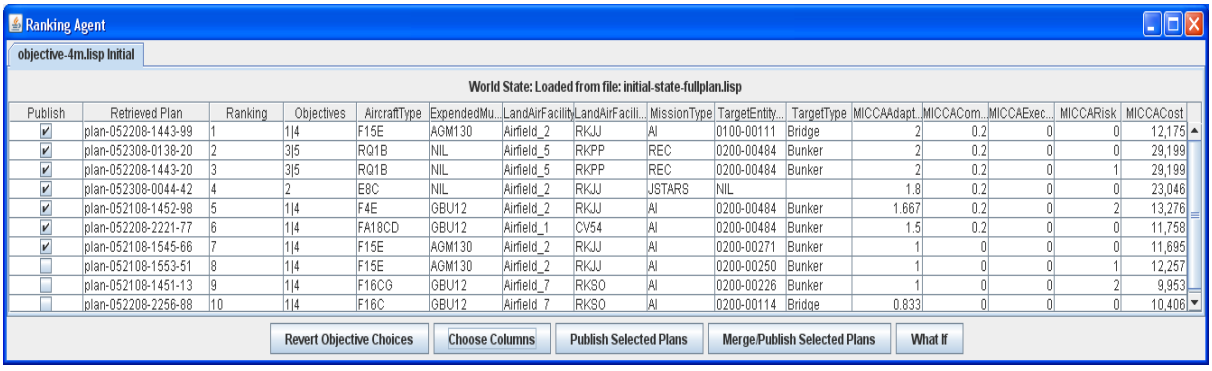

Figure 8. Ranking Agent displays the ranked Retrieved Plans for the JAGUAR domain.

This user interface also enables the user to get additional information about each of the ranked plans that summarizes how the evaluation has been performed and describes the details of the plan. This information can be displayed textually, on a map, or in a Gantt chart. Finally, the user can evaluate the potential of plans on hypothetical situations by utilizing the What-if feature. The What-if feature allows the user to create/alter the world state and trigger a new evaluation cycle with the same plans but with new conditions. Figure 9 shows how the What-if results are displayed in a different tab because they will have a different context.

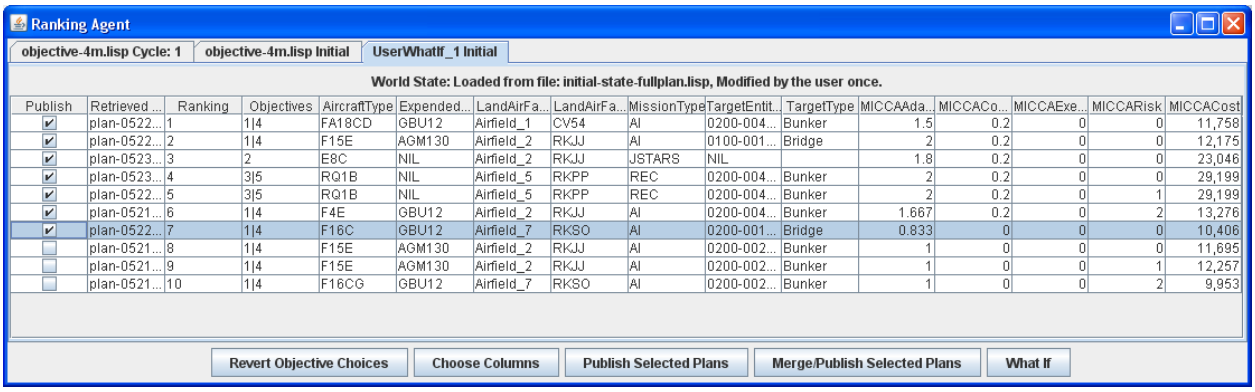

Figure 9. What-If Capability in JAGUAR creates a new world state triggering a new evaluation cycle.

3.3 Instantiation Agents (Refining candidate plans)

Once the user has selected plans for adaptation, the selected plans are aligned with the current world state. This process is carried out by the *Instantiation Agents* and serves as a preprocessing step for the adaptation cycle. (See Figure 5)

The function of the instantiation agents is to map parts of the historic objective(s) to the current objective(s) and edit the historic plan to reflect that mapping. Also in the absence of a case based planner, instantiation agents act as a simple planner by providing a technique to merge elements from multiple historic plans together to produce a candidate plan that covers most of the current objectives. These two tasks are supported by two instantiation agents: the *Editing Agent* and the *Merger Agent*.

3.3.1 Editing Agent

This agent maps the current objectives to the goals of each retrieved historic plan. It assumes that each retrieved historic plan (which can consist of multiple tasks) can be mapped to one or more of the current objectives. This mapping can be automatically computed by objective similarity matching, or manually defined by a user through the Ranking Agent GUI. If a single task in a retrieved historic plan can be mapped to multiple current objectives, the editing agent will create multiple copies of the plan (called Candidate Plans), one for each potential objective mapping.

Once the objective mapping has been established, the retrieved historic plans are translated into current Candidate Plans through the use of three transformations:

1) *Prune Unneeded Goals:* Any tasks in the historical plan that satisfy objectives that are not present in the current objective set are pruned. Since the plan is represented as a HTN, the pruning is simple; the entire node corresponding to the task to be pruned is removed from the tree. This transformation is domain-independent.

2) *Update Resources:* Resources such as actors (aircraft for example), sensors, munitions, and airbases may be identified explicitly in the objectives and defined explicitly in the plan. This transformation searches the current world state for the resources used in the historical plan. If an exact match is not found, a similarity metric is used to find a replacement that is available in the current world state. This transformation has domain specific components. The set of resources to be updated and the similarity metric is determined through domain specific code. In addition, the search algorithm to locate a replacement resource can be overridden by domain specific code to provide a directed search tailored to address specific details of the domain. Since there can be multiple potential resources in the current world state that could be used to update a historic plan, this transformation can produce multiple potential candidate plans, each of which can be evaluated and adapted independently by the other agents. In Section 3.3.1.1 we provide a JAGUAR example and describe the details of how candidate plans are produced and how resources are selected.
3) *Update Goal:* The goal activity of the plan is the activity that directly satisfies the objective, such as the strike activity for a "strike target" objective. Certain parameters of this goal can also be modified from historical values to match the requirements of the current world state. Details about which properties of the goal can be edited are left up to domain specific code. In the JAGUAR domain, for example, we allow the editing agent to modify the location of the goal in the historical candidate plan, thus allowing MICCA to use a historic plan that has a goal of the same type as the current objective, but at a different location. An adaptation agent will later modify the plan to route the actor to the new location.

The output of the Editing agent is a candidate plan (or a set of candidate plans). These candidate plans may not be executable yet, and may violate constraints such as temporal or spatial constraints. For example if the current objective is "Travel from A to C" and the historic objective was "Travel from A to B" the instantiation agent will replace B's in the plan with C's, provided that the domain specific Update Goal procedure allows this modification. Note that not every instance of B has to be replaced by C. Hierarchical plan structure, namely the decomposition tree, is used to identify the subtree related to the historic goal "Travel from A to B" and to replace all instances of B in the subtree with C. Details about how to get from A to C are not handled by this editing agent; it simply substitutes C for B. Other adaptation agents such as the *Planning Agent* will handle re-computing the route from A to C

3.3.1.1 Instantiation in the JAGUAR Domain

While instantiation in Rovers proved to be fairly simple and is described briefly in Section 4.0 of this report, instantiation in the JAGUAR domain presented a variety of challenges. For the JAGUAR domain, the Editing agent makes use of the historical plan case base to determine an appropriate resource for substitution. The typical JAGUAR world state can consist of many resources suitable for a specific task and objective, so constraining the choice is a helpful tool. The *JAGUAR Editing agent* first searches for instances of an aircraft of the same type, based at the same airfield, and carrying the same type of store (munitions or sensor) as was used in the historical plan. If no such exact match is found, a case base query is performed that looks for missions of the same type, and if there is a target, the same type of target (though not necessarily the exact same target instance). The matching missions are ranked by similarity to a set of other case features including the aircraft type used in the original historic plan, the munition or sensor

being used, the originating airfield, and the exact target entity. Thus, this editing agent will only propose candidate plans with specific aircraft type/munition type pairing that was used before in a historic plan and stored in the case base. This editing agent does a double pass search through the ranked list of returned cases, first looking for an exact match of the aircraft type and a different munition type. If no exact match is available in the current world state, the second search pass will try to use a different aircraft type that was used in a returned historic plan along with the munition type that was used in that historic plan. If there are still no possible instantiations in the current world state, this editing agent will give up and produce no candidate plan. Although not implemented, an additional search step could be added that searches aircraft and munition model capability definitions to find a new aircraft and munition pair that was not used before and could satisfy the objective.

3.3.2 Merger Agent

When no single retrieved plan achieves all of the current objectives, but a collection of retrieved plans can satisfy a subset of objectives, the Merger Agent is used to combine several retrieved plans in order to produce a full candidate plan that achieves more of the desired objectives. Merging plans in an effective way generally requires a complex reasoner, such as a case based planner. MICCA does not have this technology, and instead we have approximated this process by using a multi-processing and mixed initiative approach. We assume that in the first cycle MICCA will revise candidate plans that can achieve only a subset of the current objectives. After the revision and evaluation cycle, the user will pick the most promising revised plan(s) to be merged into a new candidate plan in order to cover as much of the entire set of new objectives as possible. The user can choose multiple plans that cover a single objective, which would give the merger agent multiple options for merging, and thus a decision to make. The merger agent will produce multiple merged candidate plans if there are multiple options to cover a specific objective, up to a threshold number of plans. In order to produce candidate plans that are as different as possible, the merge algorithm will look at how often each chosen retrieved plan has been used to cover an objective in the set of output merged candidate plans, and choose the retrieved plan that has been used the least.

Although the plans that are being merged are revised to work in the current (not historic) state, the combined plan might require further revisions to ensure coherence of the pieces. For example, the temporal order on the objectives may need to be revised. Thus, after merging the plans, which in this case means concatenating them in the correct order, removing duplications and combining the objectives, a revision cycle will be triggered.

In the case when an objective is still uncovered because none of the historical plans satisfy this objective, a message is sent to the user that there remains an unsatisfied objective. This information is provided to the user in the Ranking Agent interface. Figure 10 is an example that lists all of the objectives that are satisfied by the revised plans. Note, in this case, there are nine objectives to be satisfied and the information in the Objectives column of the Ranking Agent GUI indicates that for each of the Revised COAs (RCOAs) only eight of these objectives could be satisfied.

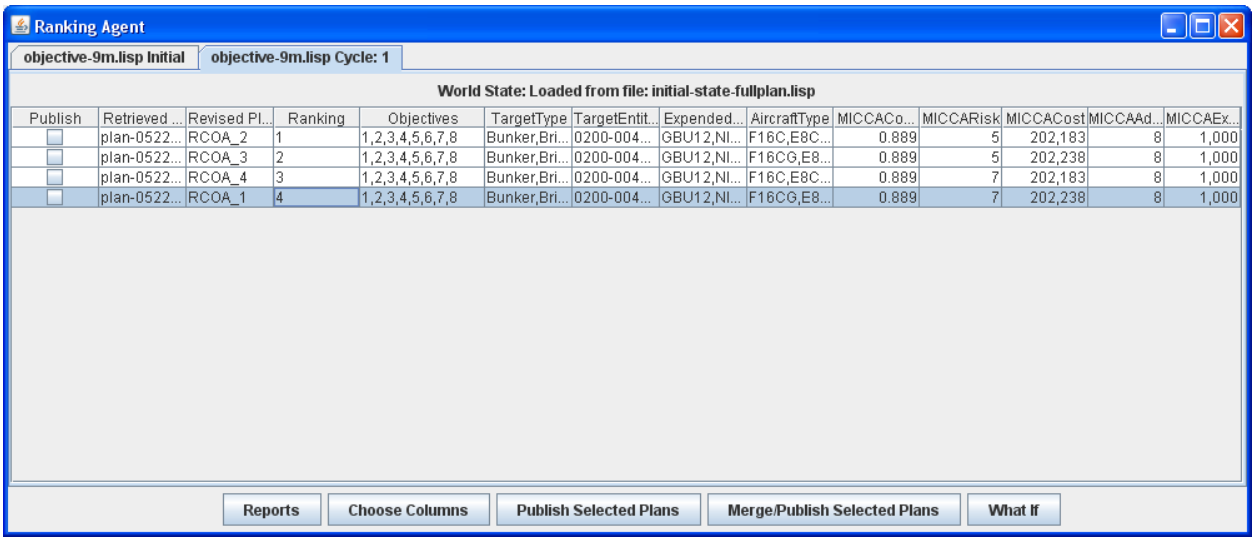

Figure 10. The "Objectives" column informs the user about covered objectives in the revised COAs.

3.4 Adaptation Agents

Several types of adaptation agents might be required to repair a candidate plan. These will include adaptation agents that are specialized in: planning, temporal reasoning, as well as specialized agents that can be designed to handle domain-specific issues, such as route planning, task priorities, risks, etc.

Agent interaction is managed through a meta-critic agent that that is called the *Coordination Agent*. Once the Coordination Agent receives input from the Instantiation Agents it will issue a Revision Request to the adaptation agents. The reasoning behind how/when to assign an adaptation agent to a specific task is dictated by the coordination policy preferred by the user (See Section 3.5) and the adaptation tasks required by the domain. An adaptation agent will respond to a Revision Request with a Revision Response. Depending on the request type, the agent will either operate on new conflicts that were already in the candidate plan (or introduced during other revision cycles) or produce a different solution for a modified set of constraints.

All adaptation agents share common data structures and the same interface. This is captured in the base class which all agents inherit from. The data structures of the base class are template-based allowing each agent to be differentiated as needed. All adaptation agents publish an *AdaptationCapability object* when they register to the blackboard. This is to inform other agents about what kind of repairs the adaptation agent can do such as CAUSAL and TEMPORAL fixes. Each agent has a *taskQueue* for RevisionRequest objects that are posted on the blackboard for the agent. As more of these objects are posted, a thread adds them into the queue. The agent will continue to process those requests unless the queue is empty.

Adaptation agents keep track of their internal states to prevent loops and to enable backtracking if necessary. For every revision response the agent will store whatever internal state information

is necessary to produce an alternative response when requested. Each agent might need to store different kinds of information.

The basic functions, such as starting the agent and adding/removing revision requests from the tasks, are implemented in the base class. However, every adaptation agent has to override the *generateNextRevisionResponse* function for creating the revision response for a new request.

3.4.1 Plan Adaptation Agent

There can be several plan adaptation agents ranging from fully domain independent to domain specific. In MICCA we have designed planning agents that are semi domain independent (e.g., developed to support the Rovers domain) and planning agents that are domain specific (e.g., developed to support the JAGUAR domain). The design for both of these types of adaptation agents is based on the assumption that the plans are hierarchical. Figure 11 is an example of a hierarchical plan for performing an Air Interdiction (AI) mission against a particular target, e.g., TGT_60.

The Plan adaptation agents in MICCA are all based on HTN (Hierarchical Task Network) planning where objectives are specified as high-level tasks to be accomplished and high-level tasks are recursively decomposed down to primitive tasks.

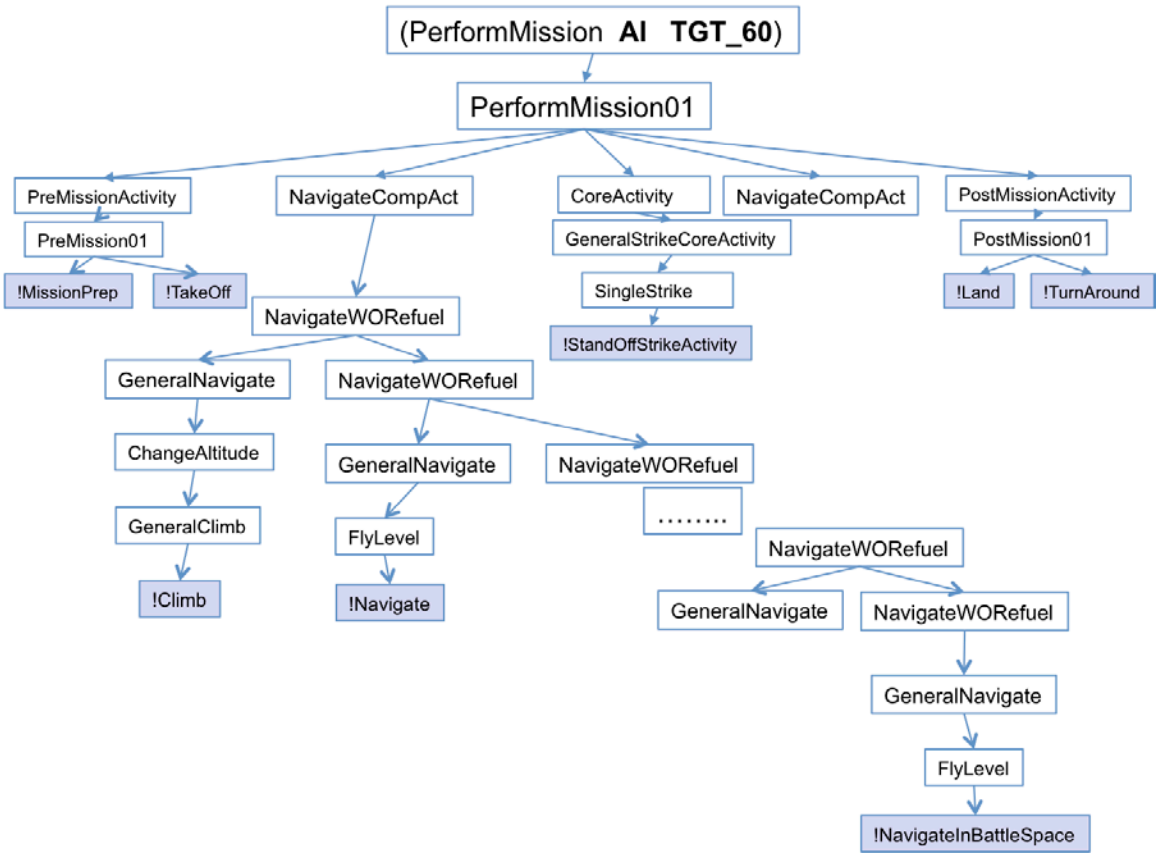

Figure 11. An example hierarchical plan in the JAGUAR domain where the top level task is to perform an Air Interdiction (AI) mission.

A plan generally contains a number of causal relationships between the tasks. These causal relationships, if violated, will need to be fixed during the adaptation process. The general approach used to fix causal relationships is:

- Find broken dependencies
  - Find all actions that fail, i.e., unsatisfied preconditions.
- Fix broken dependencies locally to preserve the plan
  - E.g., call a planner to fix the broken link.
  - Make domain independent fixes: Powered by HTN planner JSHOP.

The two main tasks of the Plan adaptation agents are: (1) Dependency analysis and (2) Plan repair. The level of domain independence depends on the type of algorithm used in these two steps.

In MICCA all of our plan adaptation agents have a domain-specific dependency analysis step, i.e., the designer of the algorithm hard codes the dependencies to be verified. A domain independent approach would involve the analysis of the HTN domain models to extract the dependencies. In our research, we have implemented two plan repair strategies. For the Rovers domain we have utilized the JSHOP HTN planner to create missing plan segments, or do repair (by means of re-planning the minimum plan subtree). For the JAGUAR domain, the agent utilizes the hierarchical structure of the plans to extract necessary information and then adapts the plans without calling the HTN planner, so we had to implement a suite of adaptation techniques in this domain. Plan repair using a domain independent algorithm such as HotRide [14] is also possible. This is particularly useful when a generative planning approach is the preferred approach. It is less important in the MICCA research because we also use case base reasoning technology to support candidate plan creation and adaptation.

3.4.2 Temporal Adaptation Agent

The goal of this agent is to compute the start and end interval for each task in the plan. This agent is designed to be domain independent. This agent requires several domain specific inputs such as: parametric temporal constraints, and a knowledge base with domain specific modeling information that can influence how the constraints are implemented. For example, in the JAGUAR domain, the speed of an aircraft will affect the start and end times of tasks.

The general approach used in MICCA for temporal adaptation is to remove all historical start and end times from a plan and then compile a *temporal constraint graph* (consisting of nodes representing events or absolute time points and weighted directed edges representing the temporal distance between nodes in seconds). We then compute the distance between all nodes and a special timeline node that marks the beginning of the timeline (any time point before the current time is acceptable for specifying the beginning of the timeline). If the distance of any node to itself is negative then the set of constraints represented by the constraint graph is unsolvable. Otherwise, the distance to/from the timeline node gives the temporal window for the event represented by the node.

As shown in Figure 12, a hierarchical plan, a current world state, an objective statement and a reference timeline point are used to compile a constraint graph. Figure 13 demonstrates some of the steps of the algorithm for a single action plan.

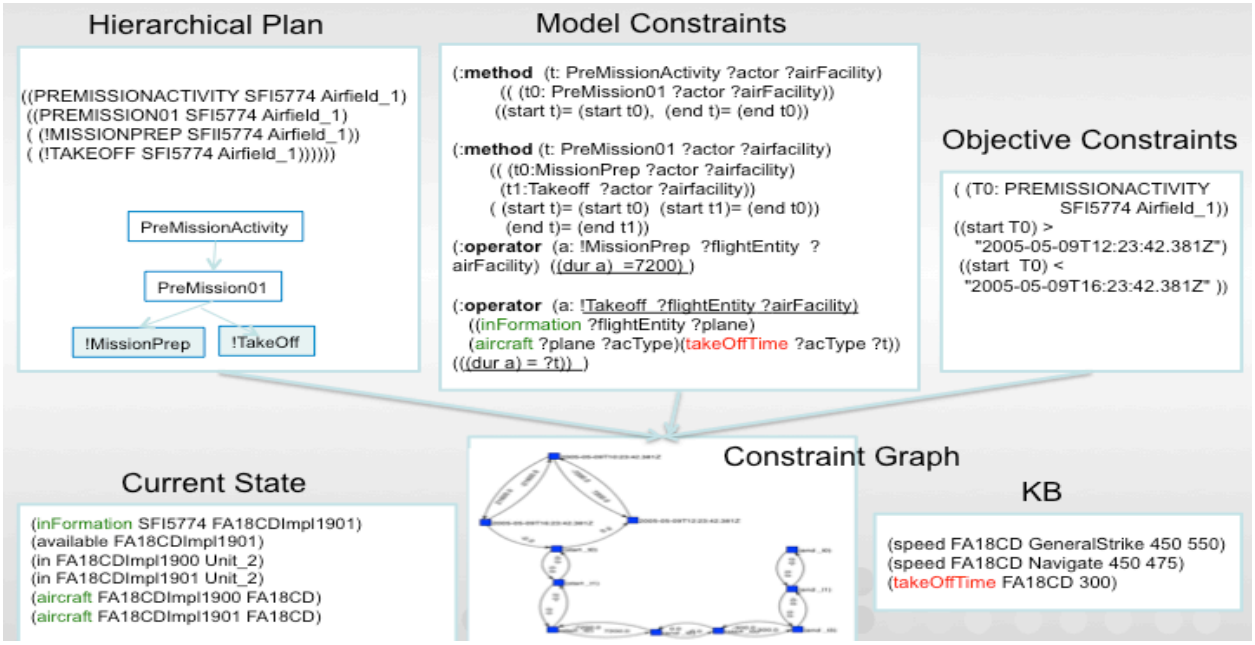

Figure 12. Inputs required for compiling the constraint graph representing all temporal constraints related to a given plan.

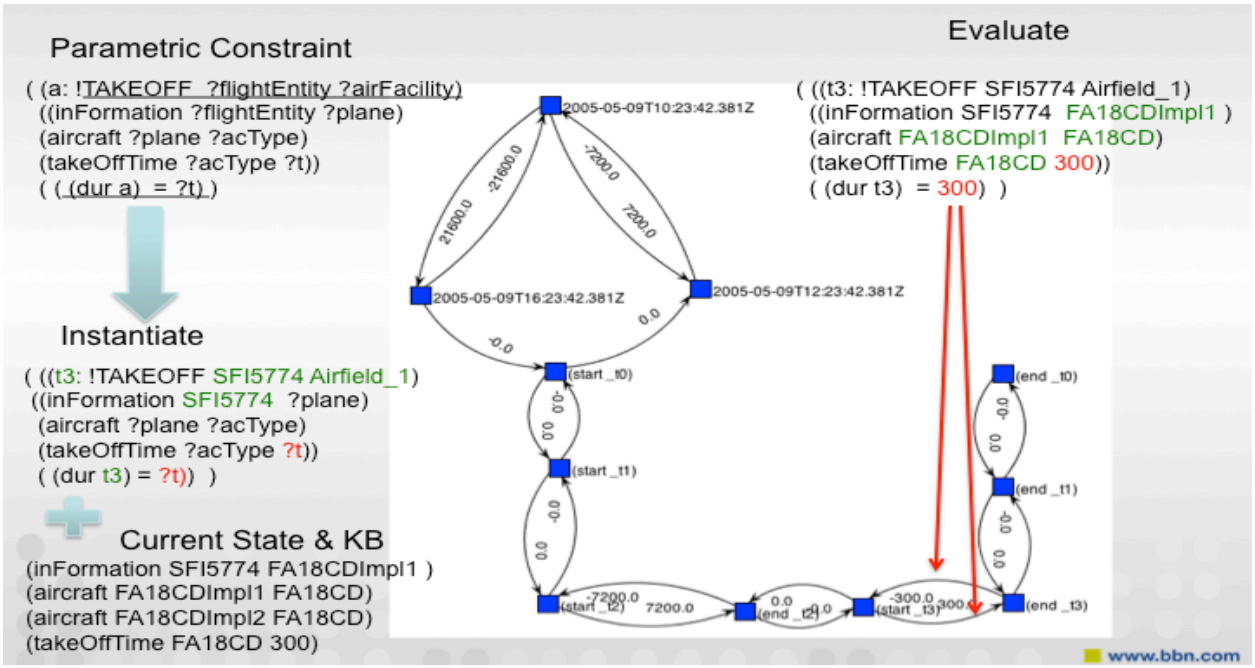

Figure 13. Example demonstrating some of the steps of constraint graph compilation for a single temporal constraint.

To solve for the shortest path problems we employ the Floyd-Warshall algorithm [15] which runs in $O(n^3)$ time where n is the number of nodes in the graph. Several highly effective optimizations are possible for both reducing the size of the constraint graph and the processing time of the graph, such as:

- Reduce the number of nodes in the graph by keeping track of equality relationships between nodes. This can be done either as a post-processing step where the equivalent nodes are merged or at the graph creation step. Careful bookkeeping operations are needed to handle the references to the original nodes.
- Analyze the constraint graph to identify connected components and then process the components separately thereby reducing the complexity of the temporal window computation. This step can also be parallelized.
- The temporal agent keeps track of its internal state when processing a revision request. After the creation of the temporal graphs the agent save the graph and adjust it incrementally to reflect the new constraints in the new request. This will reduce the amount of time spent on creating the graph.

The current implementation of MICCA does not do any optimizations. Our major assumptions for ensuring tractability are: 1) the temporal constraints are not-disjunctive thus can be solved in polynomial time, and 2) there is a unique temporal constraint per decomposition, i.e., the list of subtasks ensures that the hierarchical plan has a unique derivation tree.

The Temporal Agent's consistency checking is not incremental. If the set of constraints is consistent, then the shortest path algorithm will compute the distance between all nodes and provide the temporal window for each temporal variable. However, if the set of constraints is *not* consistent, then the algorithm simply will return the result that no solution exists. It will not specify which constraint is conflicting with others and it will also not produce any temporal windows for variables that are free of conflict. This is because the algorithm evaluates the constraints as a whole.

To enable conflict detection and partial solution generation we have developed an Analyzer. Because we assume that our constraint knowledge base contains conflict free constraints, the major assumption in the analyzer is that: "any inconsistency is introduced by the objective constraints". This is reasonable because constraints (in the domains we have used to date) in the knowledge base often come from procedures that are fixed. The analyzer computes the plan constraints and then incrementally adds objective constraints. If at any point the set becomes inconsistent the analyzer will stop and mark the latest addition as conflicting.

This approach may require consistency-checking depending on the number of objective constraints. Given the computational cost of consistency checking, which is $O(n^3)$ where n is the number of events, this is not practical. To reduce the complexity of the problem we exploit the hierarchical nature of the plans. After the graph for plan constraints is generated (see Figure 14 for an example), we compute and extract the temporal windows for the milestones of top-level tasks only. This creates an always-solvable set of constraints, and creates events no more than a small factor of the objective tasks (Start, End, ObjectiveWindows, and FlightWindows). The smaller graph in Figure 14 is used to support incremental consistency checking in identifying a conflict.

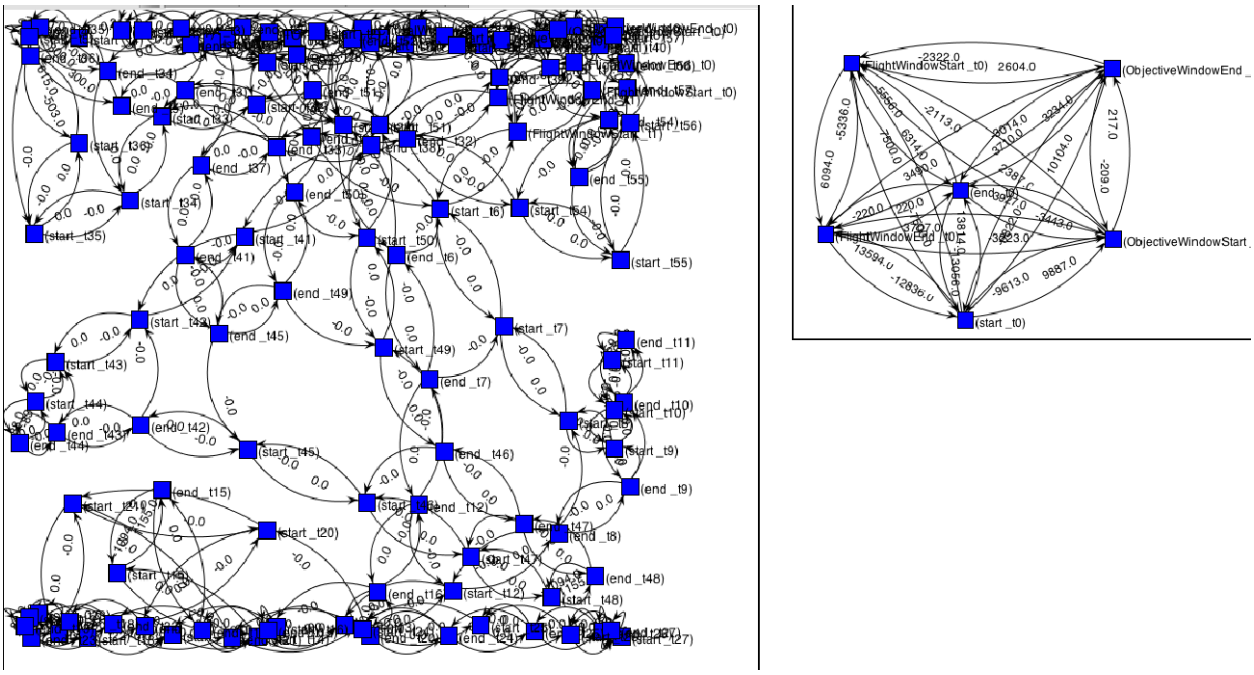

Figure 14. The complex temporal constraint graph (on left) can be reduced into a simple graph with top-level variables only exploiting the hierarchical nature of the plans.

Note that the insertion order of the objectives, when checking conflicts, might yield different results. Once again we assume that the conflicts can be introduced by unary constraints that place a deadline on any temporal variable. Thus we do not analyze the effect of binary (relational) constraints on consistency. For unary constraints, we use objective priorities (if available) to impose an ordering on the constraint. The set of conflicting constraints (if any) is stored in the Revision Response so that the coordination agent can present it to the user or use it for auto de-confliction.

3.5 Coordination Agent

Given a candidate plan which might need multiple adaptations possibly by multiple agents, we need to address the issue of how the adaptation agents will interact with each other to produce coherent plans. One approach is to let the agents with different capabilities operate in parallel, i.e., each agent will propose a repair independently from the current activities of other agents. However, this approach can waste repair cycles if agents suggest conflicting revisions. Furthermore, merging the suggested repairs is a hard problem. The second approach is to pick one adaptation agent to work on the plan in a given cycle, fix everything it can and then hand it over to another agent. This second approach necessitates a *meta critic agent* that is responsible for imposing an order on the adaptation agents.

In MICCA we have implemented the second approach and developed a Coordination agent to control the interaction between adaptation agents. The function of the Coordination Agent is to:

- keep track of adaptation agents and their capabilities
  - Adaptation agents will post capabilities on the blackboard.
- keep track of domains and their needs
  - e.g., the Rovers domain will need causal and path planning; the JAGUAR domain will need causal, temporal and resource management.
  - The human will declare the kind of adaptations necessary for a domain.
- implement a set of agent interaction policies, including:
  - A basic policy, which is analogous to depth first search, where a predefined order on the adaptation agents is employed to decide which agent will work next on the current plan. If an agent reports a dead end, the Coordination Agent will request help for de-confliction.
  - Complex interaction policies will be used for implementing directed back jumping. Encoding search heuristics can also be defined to support policies.
- implement a set of human interaction policies to:
  - Define triggers for user input.
  - Determine the frequency of user input request.

Figure 5 shows the interaction protocol between the adaptation agents and the coordination agents. It also displays interaction of the coordination agent with other MICCA agents such as the instantiation agents. The coordination agent also communicates with the human operator, asking for help when there is a conflict and using policies that the user has specified to support adaptation.

The Coordination agent executes an adaptation policy which is a sequence of capability declarations from the adaptation agents. The agents are ordered with respect to their dependencies, i.e., no agent that has a capability dependency will appear before the agents that satisfy the dependency. This policy can be executed in a serial or a parallel manner. The execution will depend on the policy executor that is implemented. In MICCA we have implemented a serial policy execution to propagate the candidate plan through the adaptation agents in the same order as they are described in the policy. If at any point a conflict is reported the executor can either take automated action or ask the user for help. Depending on the de-confliction method (temporal or causal), the execution jumps back to the first agent in the policy that has the capability to interpret the modification.

The decision to ask for user help during de-confliction or enable auto-fixing is controlled by a configuration parameter, which limits the number of auto fix attempts an executor will do before asking the user for help. Currently the coordination agent is capable of auto-fixing temporal constraints. The approach taken is to adjust the unary constraint to the minimum acceptable value. This value is computed from the partial-solution reported in the revision response by the temporal agent. A human might choose to edit another constraint or even suggest a plan edit through the de-confliction GUI to solve the same problem. (See Section 3.6).

The user has control over the list of adaptation agents he/she wants to work with. The Adaptation Policy Setup GUI, as displayed in Figure 15 is available to enable the user to review and/or set policies for the Coordination agent. This window displays the available adaptation agents and identifies the current adaptation policy. Note that Causal and Resource adaptation

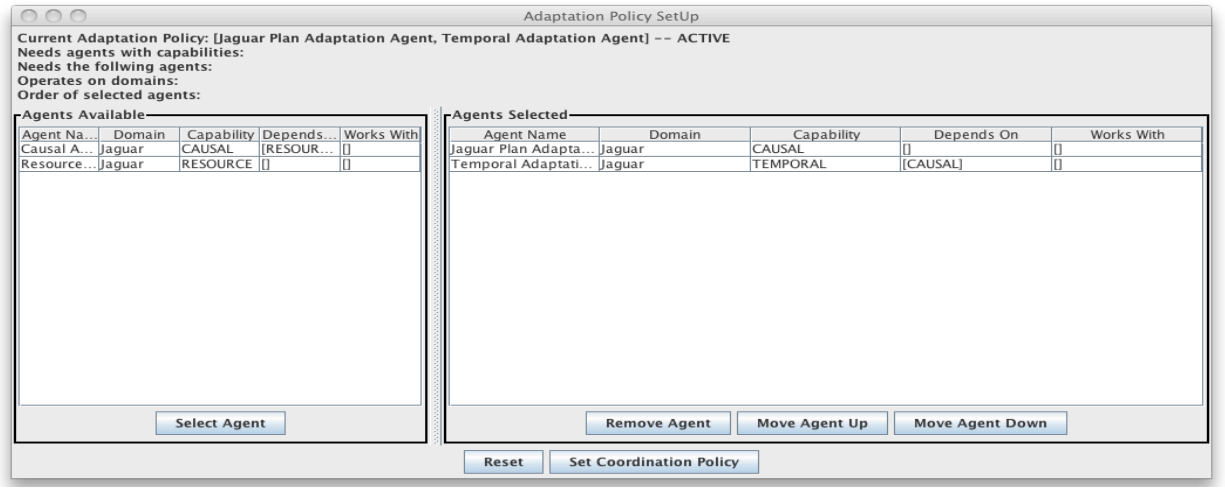

Figure 15. The Coordination Adaptation Policy GUI.

agents were not described in the previous section because these agents are simple place holders that are being used to demonstrate the support of more complex adaptation policies and dependencies. The agent and capability dependencies are also displayed to help the user pick the compatible agents. As the user adds or removes the agents from the policy through the interface, MICCA methods associated with this GUI check the coherence of the policy, i.e., check if all the required capabilities are added into the policy and/or report possible incompatibilities due to domain clashes etc. MICCA methods associated with this GUI also ensure that the selected agents are always in an order that will satisfy all dependencies. Any problems are displayed to the user on the top panel of this GUI. The user can change the existing policy by clicking on the "Set Coordination Policy" button.

3.6 Mixed Initiative

The human operator is treated as an agent within MICCA. Several GUIs have been developed during the course of this project to help the user understand MICCA options and reasoning and to enable the user to influence how both evaluation and adaptation are performed. Table 2 describes the high level user tasks and references the GUI that is available in MICCA to support the task. Because various figures are scattered throughout this document, the Figure Reference identifies the figure that contains an example of each of the interfaces. Note that some of the GUIs are not provided in the main part of this report but are instead included in Appendix E.

Table 2. User task, HCI functions and GUI tool name and references to Figures as they appear in the paper.

Task	HCI Function and/or content.	GUI Name	Figure Reference
Perform Situation Assessment	Blackboard (BB) messages about the objective, commander's intent (CI) and world state.	Various BB messages	Figure E-1
Query Case Base Results	A query tool is available to build a case base query. The results are published to the Ranking Agent GUI for review.	Ranking Agent GUI, Case Base Browser	Figure 7 Figure 8 Figure 23 Figure 28
Influence evaluation	The evaluation agent management GUI allows the user to view the Lexicographical Preference Model (LPM) that the evaluation agents will use and to make changes to this model to influence the evaluation of retrieved and/or revised plans.	Evaluation Agent management GUI	Figure 6
Create/edit evaluation agents.	The user can create new evaluation agents and/or edit existing agents.	Evaluation Agent management GUI	Figure 6
Create adaptation policy	The user can influence the order of adaptation by changing the policy used by the MICCA agents.	Adaptation Policy Setup GUI	Figure 15
Review information about plan evaluation	The user can review the rational for why retrieved or revised plans are ranked. The user can also review the plan/COA information graphically on a map or in a Gantt time display.	Ranking Agent, Plan Detail, Map Tool, Gantt display	Figure 8 Figure 17 Figure 24 Figure 25
What-If analysis	The user can make changes to the world state in anticipation of some problem.	Ranking Agent GUI, What-if World State Editor	Figure 9 Figure 18 Figure 31
Help Request	The user can provide help in resolving temporal conflicts.	De-confliction GUI	Figure 16
Publish Adapted plan/COA	The user can generate various reports from the Ranking Agent GUI (Figure E-2) to determine what plan/COA should be published to the BB and can then select one or more plans and publish them to the BB for usage by an external plan execution tool.	Various Reports, Ranking Agent GUI	Figure 10 Figure 29 Figure 30 Figure E-2
Load Script, previously saved experiments, and set up windows	Predefined scripts can be loaded through the Experiment Control Center. This GUI lets the user decide what agents to use and what GUIs to display. The user can also save and reload experiments with this GUI.	MICCA Experiment Control Center.	Figure E-3

As a mixed-initiative system, MICCA offers the human operator many opportunities to participate in and influence the behavior of the system. For example, the Coordination Agent will generate help requests to the user when one of the adaptation agents reports a problem. An example of a request for help is displayed in Figure 16 and occurs during temporal adaptation.

The help request displays information about the current plan and highlights the conflict (the item in red in Figure 16). The elements in Green are okay and those in black are unverified. At the left panel of the GUI is the plan detail from a task perspective. The bottom panel of the screen provides the user with information about the limits for the start and end of each task. There are a variety of ways that the user can make corrections to the constraint that is highlighted in this tool. The user can also modify the interaction policy so that the MICCA coordination agent will request more or less help from the user. When less help is requested, the adaptation agents will resolve the constraints using the data available to them.

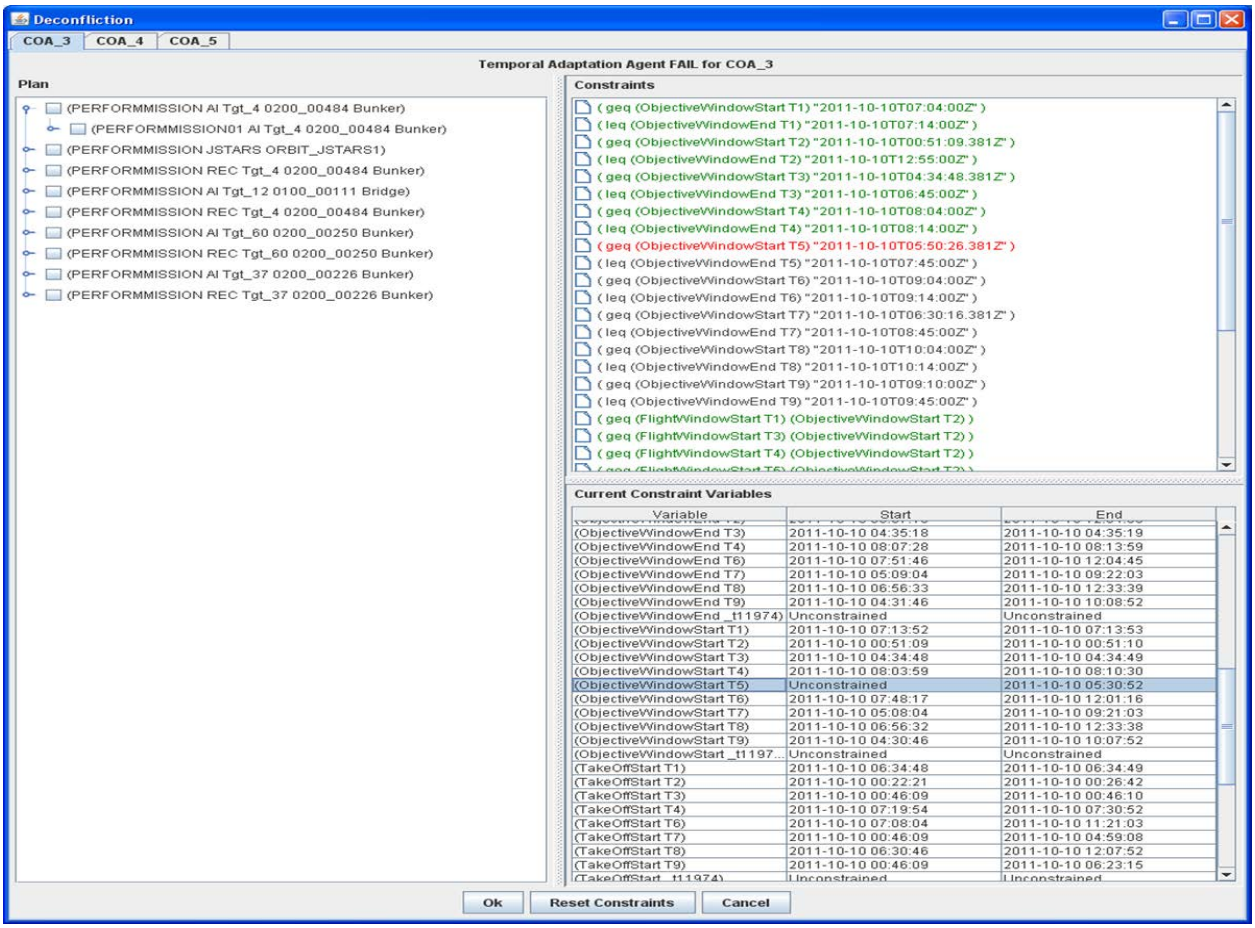

Figure 16. The user can help de-conflict the constraints by either revising the temporal constraints (right) or editing the plan (left).

The changes made by the user are turned into a RevisionRequest by the Coordination-agent. If the user has chosen a task for replanning, then the RevisionRequest's Data object will reflect that request. If the user edited any of the temporal constraints the Coordination agent will replace the original goals with new ones. The agent will also keep track of the total amount of edits to the constraints in three categories: HIGH, MID and LOW priority constraint edits. Such statistics can then be used to generate new evaluation agents.

The user can examine all edits and adaptation done on a plan by right clicking on a plan in the Ranking Agent GUI. The edits are displayed as a hierarchical tree summarizing what each agent has done and why. Easy access to varying levels of plan detail and adaptations is necessary in a mixed initiative system to let the user verify/validate the changes done on the plan. Figure 17 is an example of the plan detail explanation tree for a revised plan.

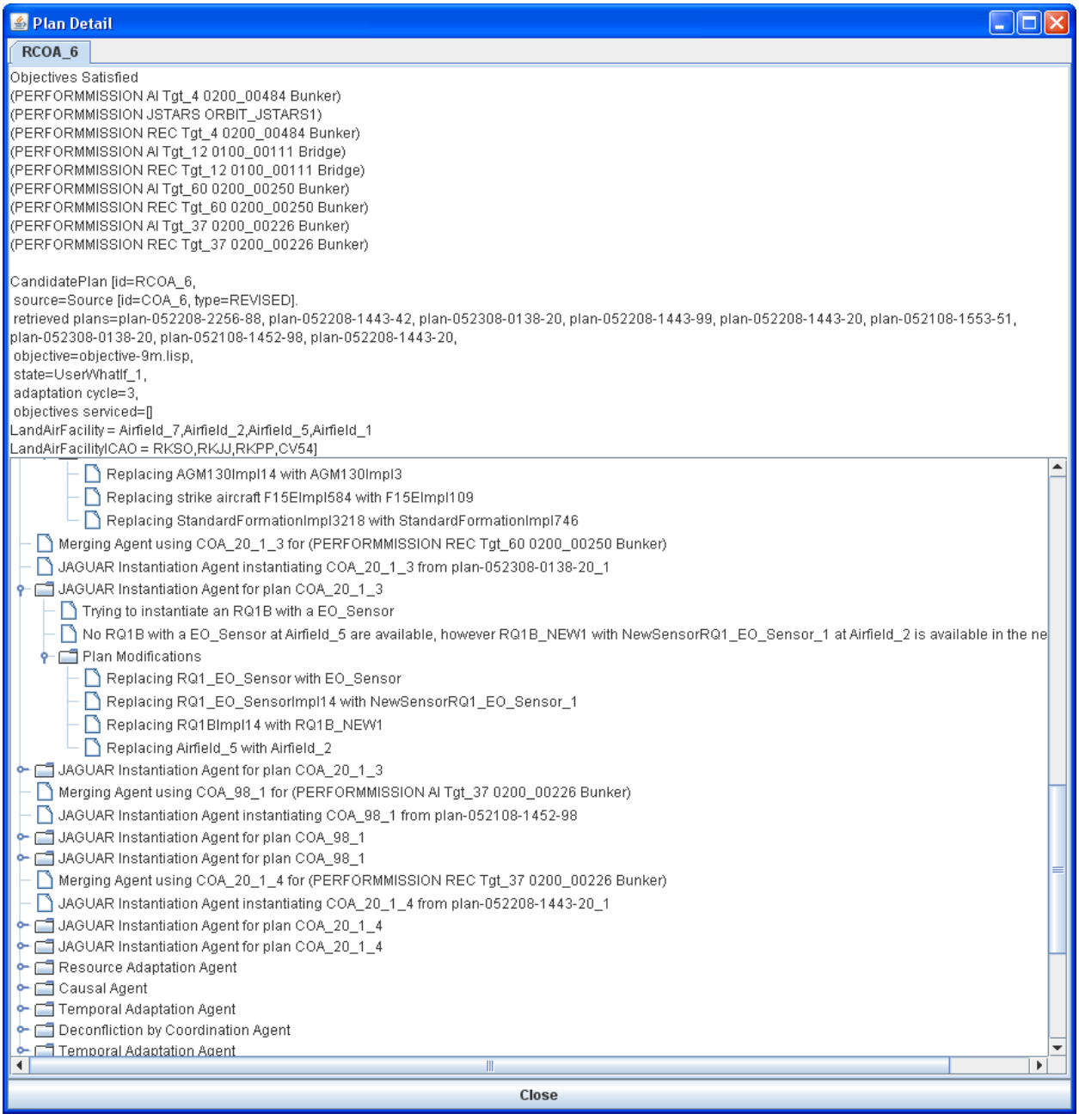

Figure 17. MICCA plan detail with information about problems and solutions.

The user can experiment with hypothetical situations using the What-If feature of MICCA. The What-If interface enables the user to view and make changes to the world state. Figure 18 displays an example of a formula that has been defined to allow the user to make coherent world state changes. In this example, the user is testing to determine if the addition of a particular aircraft could affect the scores of potential plans. In Section 4.2.2 we provide some examples of the reports that were generated by MICCA for the JAGUAR domain to further support user understandability and to help the user to determine which COA/Plan should be selected and published for continued development and/or subsequent execution.

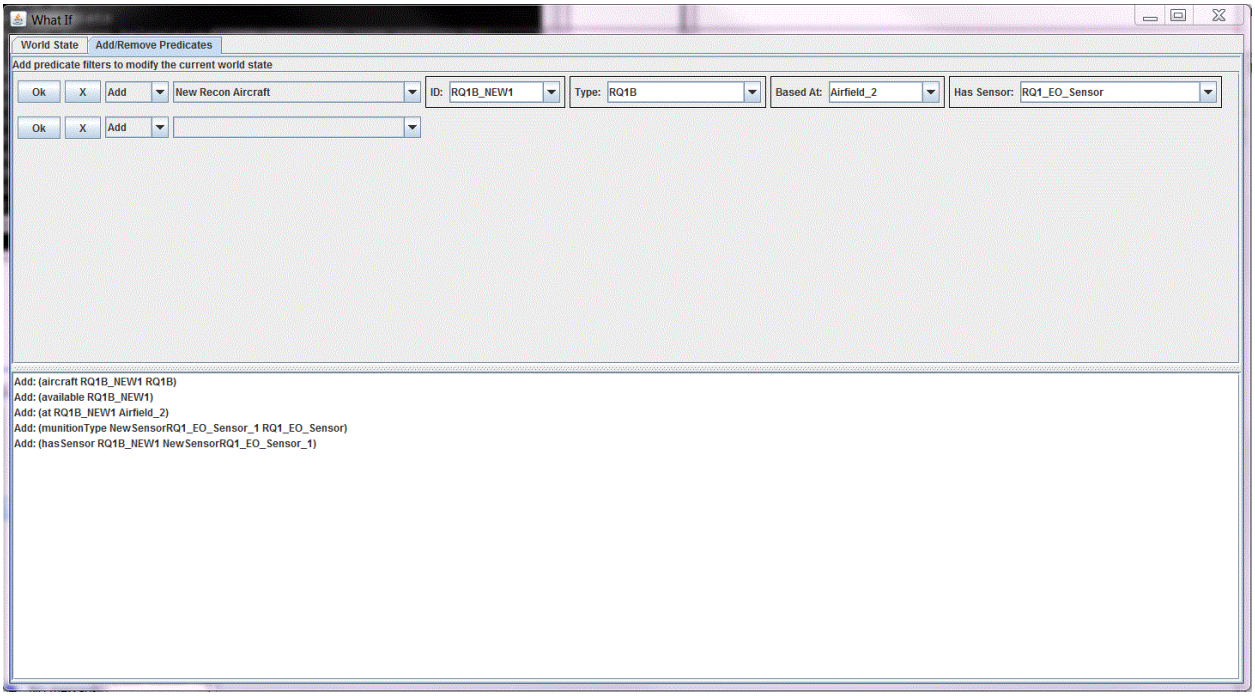

Figure 18. Modifying the World State to Support a What-If Exercise.

3.7 Input and Outputs for Agents

The inputs and outputs for each agent per task they perform are presented in Table 3. There are two types of inputs: messages passed by the blackboard (BB) and external data such as the knowledge base, which primarily resides in files or is provided as input from the user via the GUIs. Appendix B summarizes the content and structure of the knowledge bases and other data types and Appendix C describes the input/output message formats. All agents (except the Problem Definition Agent) take information about the state of the world and the current objectives as input from the BB.

Table 3. The input output requirements of MICCA agents.

Agent / Task	External Data	BB-inputs	BB-outputs
Evaluation Agents			
Cost		Candidate or Revised Plan	Scored Plan
Adaptability		Candidate or Revised Plan	Scored Plan
Risk		Candidate or Revised Plan	Scored Plan
Completeness		Candidate or Revised Plan	Scored Plan
Executability		Candidate or Revised Plan	Scored Plan
Execution Agents			
Comparison	LPM	Scored Plan	Plan Pair Comparison
Ranking		Plan Pair Comparison	Ranked Plans
Ranking/ Selection for adaptation	User selection		Candidate Plans for edit
Ranking/ Selection for adaptation	User selection		Candidate Plans for edit & merge
Ranking/ Selection for execution	User selection		Revised Plans
Instantiation			
Edit		Candidate Plans	Candidate Plans
Merge		Candidate Plans	Candidate Plans
Adaptation MetaCritic			
Coordination/ First Adaptation Cycle	Adaptation Policy	Candidate Plans	Revision Request
Coordination/ Consequent Adaptation	Adaptation Policy	Candidate Plans	Revision Request
Coordination/ De-confliction	User edits	Revision Response	Revision Request
Coordination/ Final cycle	Adaptation Policy	Revision Response	Revised Plan
Adaptation			
Planning	HTN models, Knowledge Base, Transient Knowledge Base	Revision Request	Revision Response
Temporal	Temporal Constraints, Knowledge Base	Revision Request	Revision Response
Support Agents			
Problem Definition	World State, Objective, Commander's Intent		World State Objective Commander's Intent
Browser Publishing	User Query		Retrieved Plans
Candidate Case Retriever	Cases	World State Objective	Retrieved Plans

The enabling agents provide the input for MICCA. In particular, the Problem Definition agent publishes the problem, i.e., the world state, objectives and the commander's intent. The Browser Publishing and Candidate Case Retriever agents implement alternative ways to publish the historical cases onto the blackboard as candidate COAs.

3.8 Knowledge Bases and Other Files

There are several common inputs to every evaluation and adaptation agent. In general the agents don't interact with raw input, i.e., a file containing a string in a specific format. Instead they utilize data that is refined by several MICCA parsers that can process strings that represent an objective, world state or plan. Appendix B explains the expected format for each of these inputs and provides a number of additional examples. In this section we provide a brief description of each type of data that MICCA utilizes.

3.8.1 World State

The *World state* is a list of ground (i.e., without any variables) predicates in lisp format, i.e., prefix notation with enclosing parenthesis. The world state specifies all of the resources that are available within the problem solving context. The world state can be modified during the What-if process to determine how the modified world state would affect revised plans.

3.8.2 Knowledge Base

The *knowledge base* used in MICCA contains data that is similar to the data contained in the world state and thus in the implementation, both are represented by the same data structure. The main difference between a knowledge base and a world state is that the facts about the world that are stored in the knowledge base do not tend to change quickly; such as the maximum speed of an aircraft.

Information about the World State that is likely to change due to external events is stored in a transient knowledge base. For example the operating hours of a base could change as a function of an approaching weather event.

3.8.3 Plan

MICCA plans are hierarchical which means that they are a list of decomposition trees. In addition to the hierarchical plan, the plan has a flat plan component, which is a list of all the leaves of the decomposition trees. While the flat plan can be generated from the hierarchical plan, in MICCA we pre-compute and store the flat plan separately to save computation time. This is simply because some of the agents operate on flat plans only and re-computing the flat plan in every cycle would waste time and result in decreased performance. An example of a Hierarchical Plan is provided in the following grayed box in which the top level task is PreMissionActivity which is accomplished by the task PreMission01 which consists of two simple tasks: MissionPrep and TakeOff. The numbers preceding the simple actions are the costs associated with the action. The numbers following the action names are execution order, start and end time windows.

```
((PREMISSIONACTIVITY StandardFormationImpl5774 Airfield_1)
((PREMISSION01 StandardFormationImpl5774 Airfield_1)
(7200 (!MISSIONPREP StandardFormationImpl5774 Airfield_1)  151
                        2005-05-09T12:23:42.381Z 2005-05-09T14:23:42.381Z)
(300 (!TAKEOFF StandardFormationImpl5774 Airfield_1)  152
2005-05-09T14:23:42.381Z  2005-05-09T14:28:42.381Z)))
```

An example of a Flat Plan is provided in the following grayed box which contains only the leaf actions.

```
( (7200 (!MISSIONPREP StandardFormationImpl5774 Airfield_1)  151
                        2005-05-09T12:23:42.381Z 2005-05-09T14:23:42.381Z)
  (300 (!TAKEOFF StandardFormationImpl5774 Airfield_1)  152
                        2005-05-09T14:23:42.381Z  2005-05-09T14:28:42.381Z))
```

3.8.4 HTN Models (Methods, Operators)

We have adapted the HTN domain format that is used in the SHOP planning algorithm. Specifically we have adapted the JSHOP dialect of that language because we use the java implementation of the SHOP algorithm which slightly deviates from the lisp convention. In this section we provide examples of how the components of a planning domain, e.g., methods, operators, and axioms are implemented in MICCA. Example methods, operators, and Axioms are provided in Appendix C.

3.8.5 Temporal Constraints

Temporal constraints in MICCA are quantitative. Temporal constraints for methods and operators are defined to impose either temporal ordering on the subtasks of a method or durations on the operators. Examples are provided in Appendix C.

3.8.6 Commander's Intent

This is a text file in natural language. The contents of this file can be viewed by the human operator in order to better understand the overall problem, and this data is used by the MICCA operator to manually convert the content into the "objectives file".

3.8.7 Objectives

Objectives are task lists with associated temporal constraints. The objective may contain other parameters that are of relevance to the domain, e.g., the mission type, the target instance ID, a priority value, a particular location, etc. An example Objective for the JAGUAR domain is presented here:

```
( ((T1 1 (PERFORMMISSION JSTARS ORBIT_JSTARS1)))
((geq (ObjectiveWindowStart T1) "2011-10-10T05:10:00Z")
 (leq (ObjectiveWindowEnd T1)   "2011-10-10T11:45:00Z")))
```

In this example the first element (*T1*) is the label of the task, the second element (*1*) is the number indicating the priority – one being the most important, the third is the task and finally the temporal constraints referring to the milestones of a task using its label.

3.8.8 Preference Models

The Comparison agent utilizes preference models when comparing two plans based on their scores. These preference models can be modified by the user however MICCA expects two preference models as input: one for comparing the retrieved/candidate plans and the other one for comparing the revised plans. MICCA currently supports reasoning with lexicographic preference models with numeric attributes.

3.8.9 Cases

Historical plans are stored in an XML data format that consists of five pieces of information: a unique string plan ID, the HTN plan (in the format described in Section 3.8.3), a set of case features, the world state that the plan was created for, (in the format described in Section 3.8.1), and the set of objectives the plan was created to satisfy (Section 3.8.7). The case features are a set of feature names and values that serve as descriptive meta data about a plan therefore making it readily available for searching. See Appendix B for details on the case base features.

4.0 RESULTS AND DISCUSSION

In this section we describe the two domains that we have used to support the development of MICCA. We describe some of the challenges and the approaches that we considered and implemented during the course of this research.

4.1 Rovers

To support the initial design and validation of the MICCA concept we experimented with the Rovers domain for several months. The Rovers domain was a benchmark domain in the Third International Planning Competition [16]. The choice of the domain was motivated by the following factors: (a) our existing experience with the domain; (b) the availability of HTNs; (c) the availability of a problem generator and problem sets; and (d) sufficient complexity to drive the development and testing of MICCA. In addition, the JSHOP generative planner, which we had already selected to support adaptation, could be utilized on this domain. Because the Rovers domain was being used to support the design of the system and not the ultimate application domain, our design and implementation was primarily domain independent.

The Rovers domain involves a collection of rovers that navigate a planet surface, finding samples, taking pictures and communicating them back to a lander (Figure 19). Each Rover has different capabilities that include the ability to do rock and soil analysis or capture images. This can be done in different modes and resolutions. For Rovers, we were able to leverage the existing domain models, plan and problem generators. A typical problem description for the Rovers domain is presented in Figure 20.

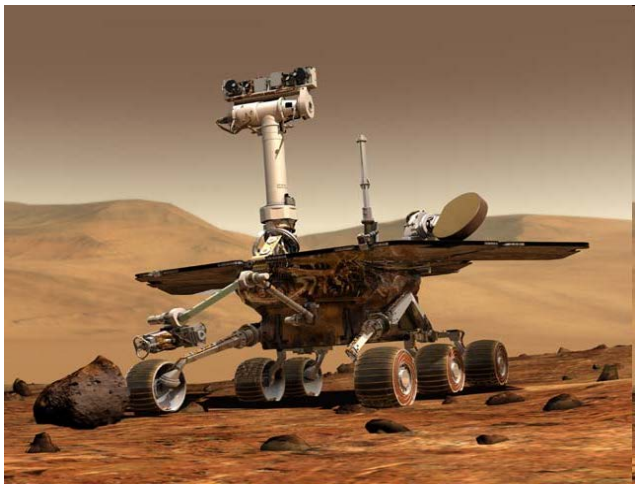

Figure 19. Example of a Rover.

During our research and development, we generated 16 random problems where:

- Entities are shared among the problems:
  A specific rover has the same capabilities in all problems.
  A specific waypoint represents the same location on the planet.

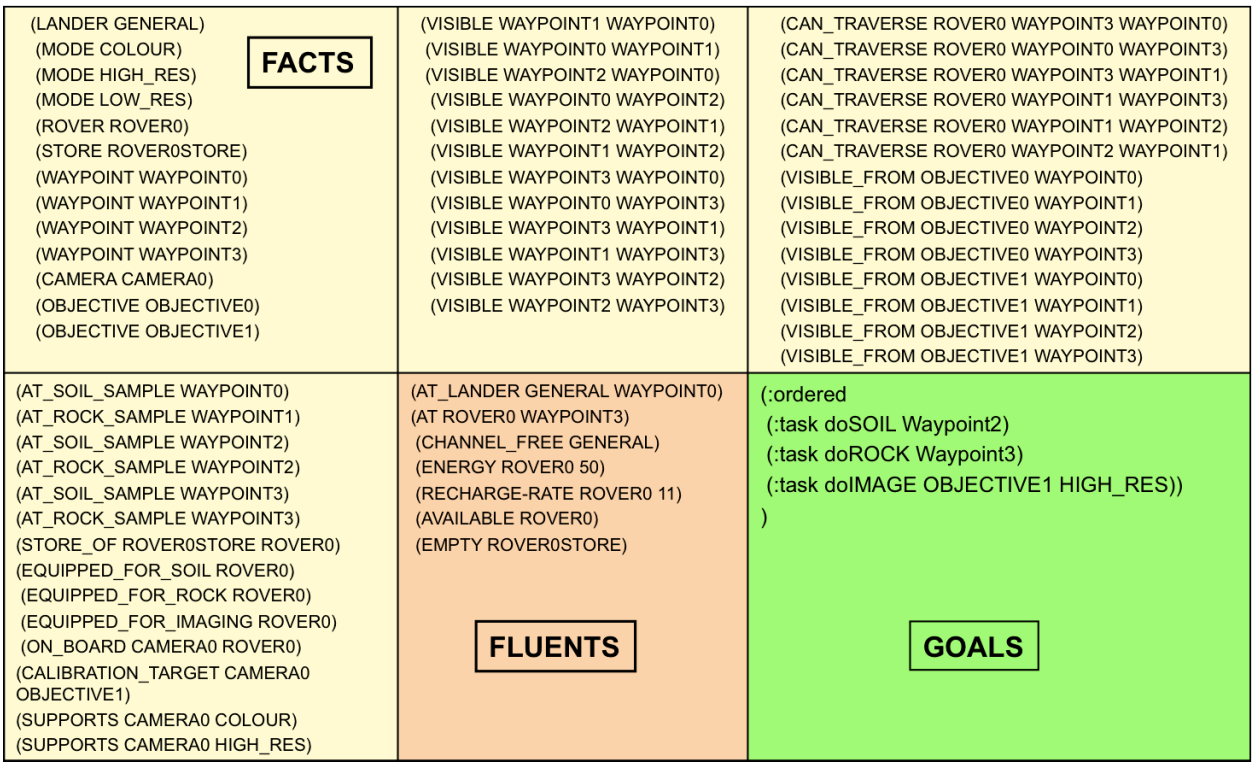

Figure 20. A typical problem definition in Rovers: yellow boxes represent the facts that are shared, the orange box contains problem specific values and finally the green box is a list of tasks to be done.

- The facts about the world state do not vary over the problems. For example, the connectivity and visibility of the waypoints are consistent over all problems.
- Fluents (a predicate that can be updated by the actions taken) of the domain are randomly assigned. For example, the locations of the rovers and lander and the energy levels of the rovers.
- The goals differ in each problem but are not disjoint.

We used the following tools to generate the use case problems:

- PDDL (Planning Domain Definition Language) problem generator – used to produce a master problem that contains all rovers, waypoints and goals, e.g., 10 rovers, 50 waypoints, and 20 goals.
- PDDL_2_SHOP problem converter – used to produce a problem that can be loaded in the SHOP2 planning system.
- Problem Partition - a new algorithm used to break a master problem into possibly overlapping case problems where fluents of a case problem might differ from the master problem.

The JSHOP HTN planner that is used in MICCA requires HTN methods and operators. We utilized the SHOP2 planning algorithm which was freely available. The original domain definition for the Rovers domain was taking advantage of the Lisp implementation by calling

specialized algorithms for shortest path computation, scheduling algorithms etc. To disable such features we simplified the Rovers HTN domain by removing the scheduling and optimization routines and energy levels. Figure 21 contains a summary of the operators and methods of the Simple Rovers domain.

<table>
<tr><th>Operators</th><th>Method Examples</th><th>Other Methods</th></tr>
<tr><td>(:operator
(!sample_soil ?x ?s ?p)
(and (at ?x ?p) (at_soil_sample ?p)
(equipped_for_soil_analysis ?x)
(store_of ?s ?x) (empty ?s)
((empty ?s) (at_soil_sample ?p))
((full ?s) (have_soil_analysis ?x ?p)
1)</td><td>(:method (DoSOIL ?goal-loc)
((store_of ?s ?rover)
(equipped_for_soil_analysis ?rover))
((navigate ?rover ?goal-loc)
(empty-store ?s ?rover)
(!sample_soil ?rover ?s ?goal-loc)
(communicate ?rover SOIL ?goal-loc ?goal-loc)))</td><td rowspan="2">(DoSOIL ?goal-loc)
(DoROCK ?goal-loc)
(DoIMAGE ?obj ?mode)
(communicate ?rover ?type ?id ?mode)
(sendData ?rover ?type ?obj ? mode)
(empty-store
(clearPath ?rover)
(navigate ?rover ?to)</td></tr>
<tr><td>(!navigate ?rover ?y ?z)
(!sample_soil ?x ?s ?p)
(!sample_rock ?x ?s ?p)
(!drop ?x ?y)
(!calibrate ?r ?i ?t ?w)
(!take_image ?r ?p ?o ?i ?m)
(!communicate_soil_data ?r ?l ?p ?x ?y)
(!communicate_rock_data ?r ?l ?p ?x ?y)
(!communicate_image_data ?r ?l ?o ? m ?x ?y)
(!!visit ?rover
(!!unVisit ?rover ?next)</td><td>(:method (navigate ?rover ?to)
((at ?rover ?from)
(can_traverse ?rover ?from ?to))
((!navigate ?rover ?from ?to)
(clearPath ?rover))
((at ?rover ?to))
((clearPath ?rover))
((at ?rover ?from)
(can_traverse ?rover ?from ?next)
(not (visited ?rover ?next)))
((!!visit ?rover ?from)
(!navigate ?rover ?from ?next)
(navigate ?rover ?to)))</td></tr>
</table>

Figure 21. Summary of the Simple Rovers domain methods and operators.

To generate the hierarchical plan for a given problem, we run SHOP2 with Simple Rover HTN methods and produce the entire decomposition tree. Figure 22 is an example of a Rover plan.

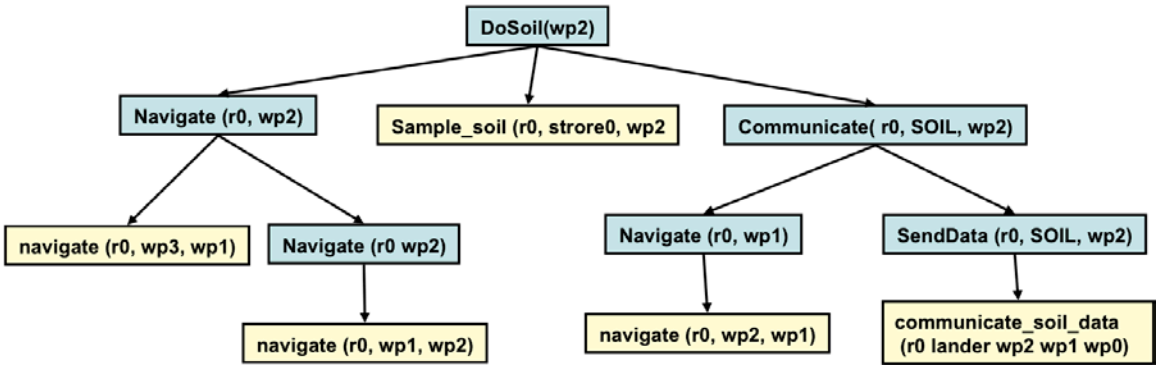

Figure 22. Visualization of a simple hierarchical Rover plan where the blue nodes are methods for composite actions and the yellow nodes are simple actions.

To support the DEEP vision, with the use of historical plans as a starting point, we imported the hierarchical plans and simple plans for all of the use cases into the case base repository. Features (attribute-value pairs) were created to support the retrieval and similarity matching of these cases from the case base. For this research we developed the following two case bases:

1. *Rover Task Case Base* with the following attributes:
   - Full goal: full goal with objective.
   - GoalType: type of task.
   - Objective: entity instance associated with task.
   - PlanID: string that describes the name of the plan.
   - Rover: agent that does the task.
   - RoverOrigin: waypoint of Rover.
2. *Rover Plan Case Base* with the following attributes:
   - HTN: xml wrapped HTN (decomposition tree).
   - Plan: xml wrapped Plan.
   - PlanID: string that describes the name of the plan.

To pass a set of HTN/Plans through the blackboard to the plan evaluation agents the user creates a query for a current goal/objective to the Rover Task Case Base which returns a list of relevant Rovers with PlanIDs who are capable of satisfying the objective. Figure 23 provides an example of the Rovers Task case base. The user then selects a PlanID and queries for the HTN/Plan from the Rover Plan Case base with the index PlanID. These plans are published onto the DEEP blackboard as retrieved plans – triggering the first evaluation cycle.

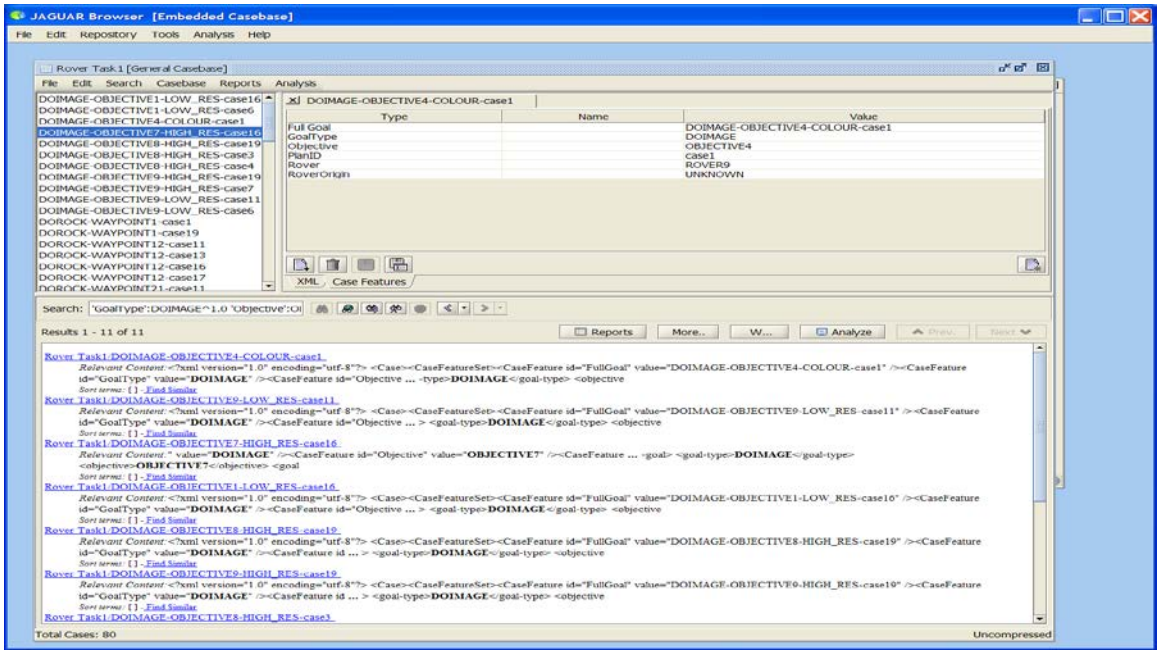

Figure 23. Rovers case base and query example.

4.1.1 Agent Configurations

Earlier versions of a subset of the agents discussed in the Methods section were developed at this stage. Namely the evaluation of the system and proof of concept demonstrations were done with the following agents: Case Retrieval, Cost, Adaptability, Comparison, Ranking, Edit, Merging, Simple Coordination and Plan Adaptation.

The Adaptability agent evaluation criteria used were:

- Retrieved Plans: Estimated effort to adapt the plan by looking at the number of related objectives, similarity of the states etc.
- Revised Plans: Adaptations preserving historic plan by keeping track of number of objectives re-planned for adapting the plan.

The Cost agent evaluation criteria used were:

- Retrieved Plans: Partial plan length.
- Revised Plans: Total plan length.

The LPM used in the Comparison agent was the same for both retrieved and revised plans:

- Adaptability score is more important than the cost.
- Higher adaptability score is preferred.
- Lower cost is preferred.

The Editing agent mapped the current objective to the goals of the historical plan and pruned unrelated ones and also did a simple find replace of old values with new ones.

4.1.2 Rover MICCA Test Examples

We tested MICCA on two levels of complexity. The first case was a simple test where MICCA was used to generate historic plans to support a *single* new objective, e.g., (do-rock waypoint12). The second test case presented the situation where there were no single historical plans that could support the new, more complex objective: two objectives and two goal types and/or three objectives and three goal types. In this situation merging is required in order to generate a resulting revised plan(s).

Our experimentation showed that the MICCA agents successfully completed all cycles resulting in improved plans that were causally sound (i.e., rovers continuously move in the space and do not jump from one location to another without any proper navigation action) and completeness (i.e., achieving all goals). To visualize the difference between candidate and revised plans we implemented a map based visualization tool that could display the movements of each Rover. An example is provided in Figure 24.

During the Rover experiments, the MICCA system was improved to support case base usage and mixed initiative interaction. In addition, the framework for the adaptation cycle was implemented. The framework includes the blackboard objects used for communicating between adaptation and coordination agents as well as with external tools and the user interface. The skeleton for the coordination and adaptation agents was also completed during the experiment.

Once our experience with Rovers verified that our framework could operate reliably, we moved to the JAGUAR domain to challenge our research.

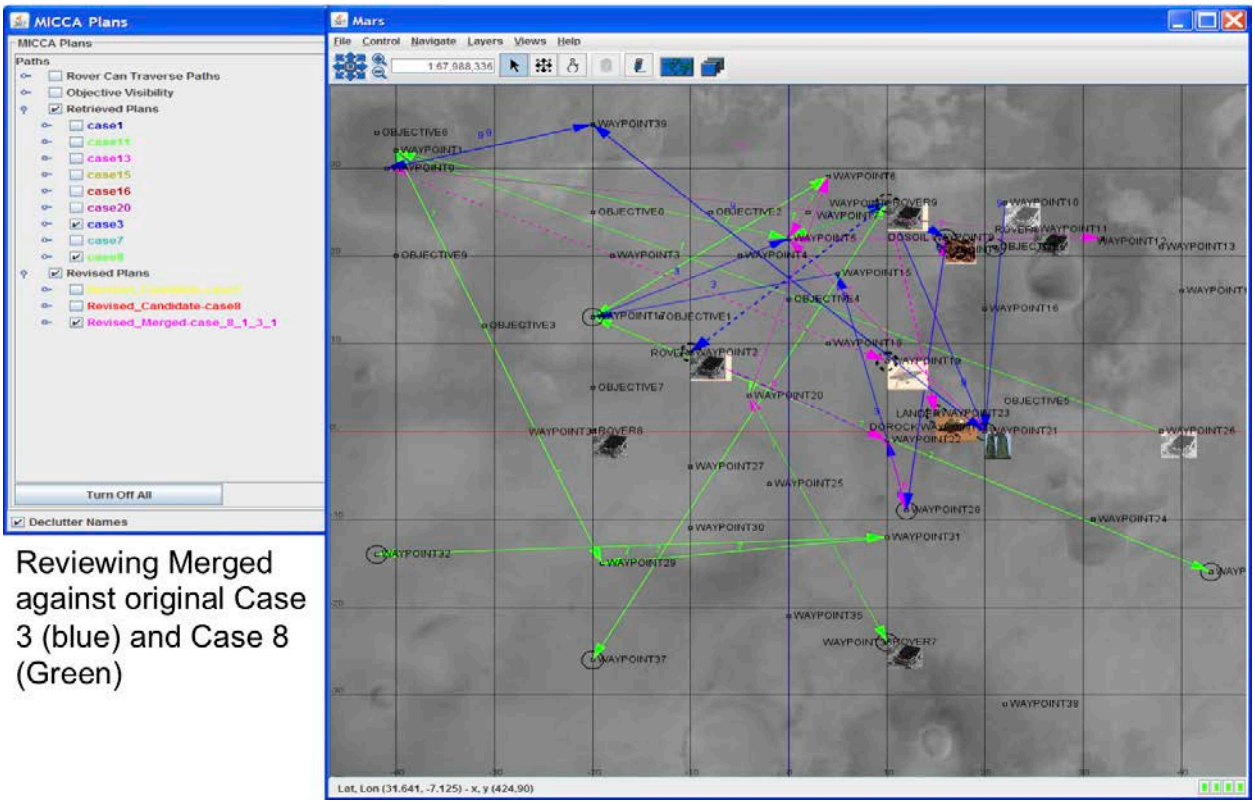

Figure 24. Visualization of revised and retrieved plans in the Rovers domain.

4.2 JAGUAR

A tool suite called JAGUAR was developed by DARPA to enhance Air Tasking Order (ATO) mission plan and execution capabilities through adaptive modeling, plan generation, execution monitoring, and dynamic replanning. The ATO is a tool that is utilized in the Air Operations Center (AOC) to support the planning and execution of a set of missions associated with achieving the commander's guidance/intent and a set of objectives for some particular theater of war. While a tactical system, the missions planned to support the ATO are associated with a set of objectives that need to be evaluated and constantly realigned with the current world state in order to be realized. Figure 25 presents a display of a set of air missions executing in parallel.

We chose the JAGUAR domain for MICCA because of the following factors: (1) data availability, (2) team member domain familiarity, (3) military relevancy, and (4) complex planning issues. Plans in the JAGUAR domain, like the Rover domain, contain an actor, in this case an aircraft with particular functional capabilities to perform certain activities. The aircraft is located at some origin and has to travel to another location in order to accomplish an objective within a specific time window – drop a weapon on a target, take a photograph of the target location, provide coverage for other missions.

In JAGUAR, as plans are executed (realized), the executed plan data is stored in a case base. The JAGUAR models and the historical plan data serve as input to MICCA. While the motivation for selecting this domain was again availability of the data –some level of domain

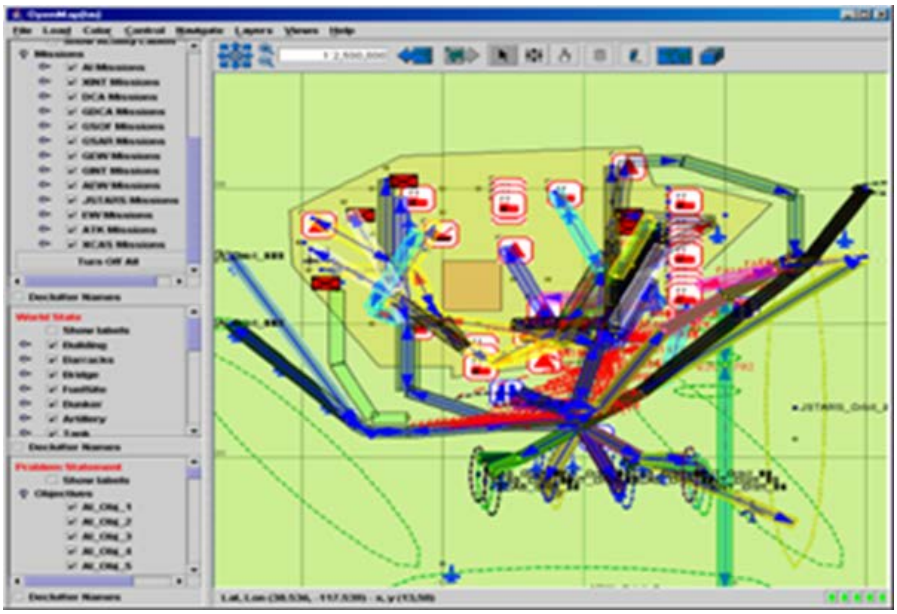

Figure 25. Visualization of the missions, routes, targets and battle space in the JAGUAR Air Mission Planning and Execution domain.

engineering was required to transform the existing JAGUAR data and models into a form that MICCA could utilize. Note that while this domain has more military relevance, it is not the ultimate application domain. However, the complexity of this domain prompted us to develop and showcase advanced features such as complex agent interactions and user and/or system customization.

To utilize JAGUAR data in MICCA we developed JAGUAR HTN models and modified the existing JAGUAR executed plan case base to include the HTN plan data. This involved domain engineering, plan extraction and finally case base creation. To scope our effort, we initially focused on only one of the ATO mission types – Air Interdiction (AI). An AI mission is an air mission that can be employed to destroy, neutralize or delay the enemy's military capability. We created a case base of AI missions and an HTN model to represent the rules and tasks for the AI mission. Over time, we developed additional HTN models for other mission types and extended the case base to include those additional mission types (AI, REC, and JSTARS).

The structures of HTN Methods and operators were extracted from the JAGUAR process models– for AI missions, REC missions and JSTARS missions only. This included **24** Methods and **14** Operators. Figure 26 displays an example of the PerformMission activity associated with the AI Mission being decomposed into sub tasks. We manually selected the predicates that appear in the process constraints and represented most of these constraints as the preconditions and effects of the methods and operators. We also manually defined **19** axioms to support the complex evaluation of conditions; such as what does it mean to be in battle space. We verified the validity of the domain by feeding it into the SHOP planner, which produced plans for simple problems.

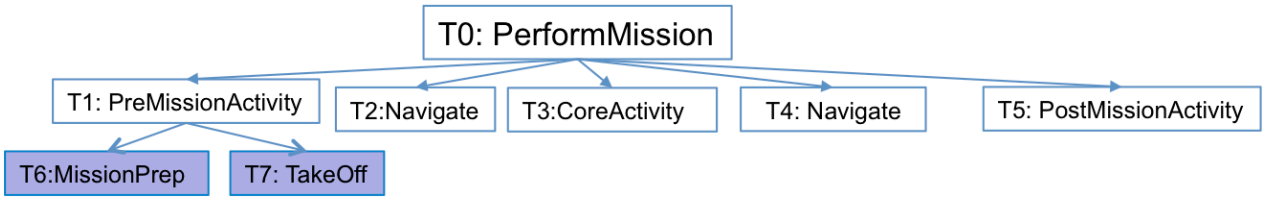

Figure 26. A method for decomposing the PerformMission task into five subtasks. This method is parametric and holds for any mission type.

The JAGUAR domain also supports very complex routes and routing procedures, which were beyond the scope and the interests of this project. To keep the project focused we simplified the routing in the JAGUAR HTNs for MICCA experimentation. For example, in our models the battle space is a polygon, defined by its corners. There are corridors which are safe entry-exit routes in-out of battle space and they are defined by three points; out, border and in. Airbase locations are outside of the battle space and we have defined at least one orbit point for each airbase. Aircraft climb and/or descend to and/or from the orbit.

Targets are points identified in the battle space and are defined as a target type, e.g., bunkers, bridges etc. For each target there is least one strike point. Aircraft can navigate to a strike point to deploy weapons (munition). Figure 27 depicts a typical navigation for an aircraft heading for the target.

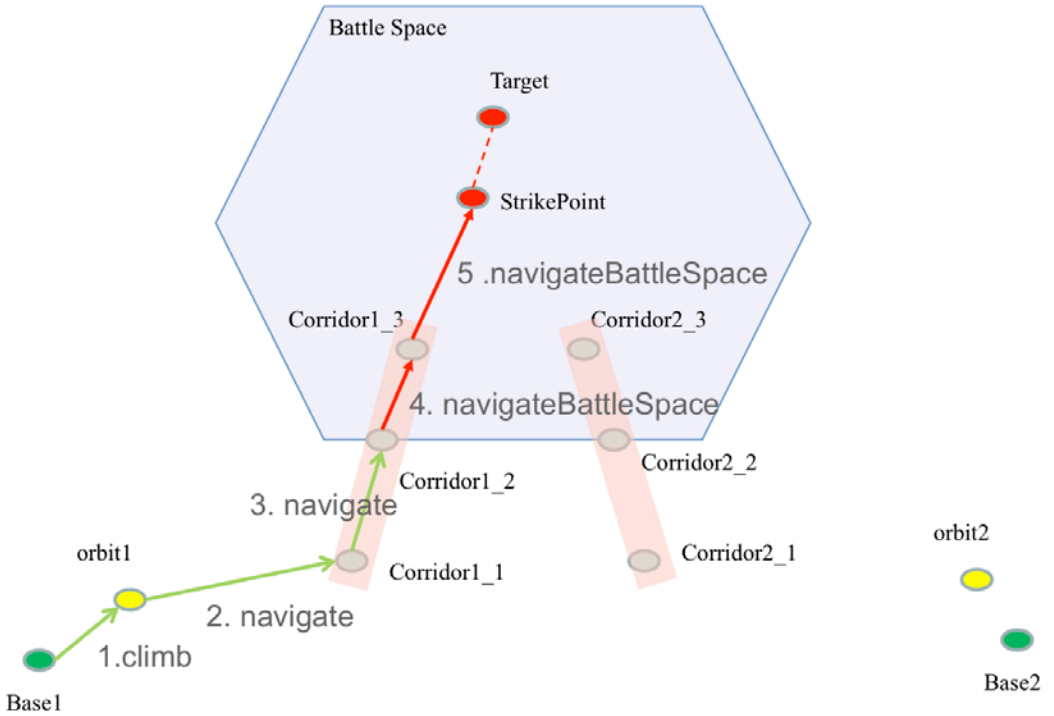

Figure 27. Simplified routing for JAGUAR as used in MICCA. The aircraft leaves Base 1 and heads toward the Target in the battle space. The navigation has several legs and is through one of the safe corridors shaded in pink.

We converted the existing JAGUAR plans to MICCA plans. This involved, format conversion (XML to SExp) and mapping of the JAGUAR plans into hierarchical plans (decomposition tree) which are consistent with HTNs. Finally we generated flat plans from the leaves of the hierarchical plans. Much of this conversion was done through the tools we developed to extract and post-process plans (See Appendix D for details).

The initial MICCA case base for the JAGUAR problem domain was populated with **2** JSTAR and **3** REC missions in addition to **19** AI missions. To create the case base we have paired extracted plans, extracted states and generated objectives through an automated and configurable process utilizing XPATH queries. Figure 28 is a snapshot of a case from the JAGUAR case base that was used for MICCA testing.

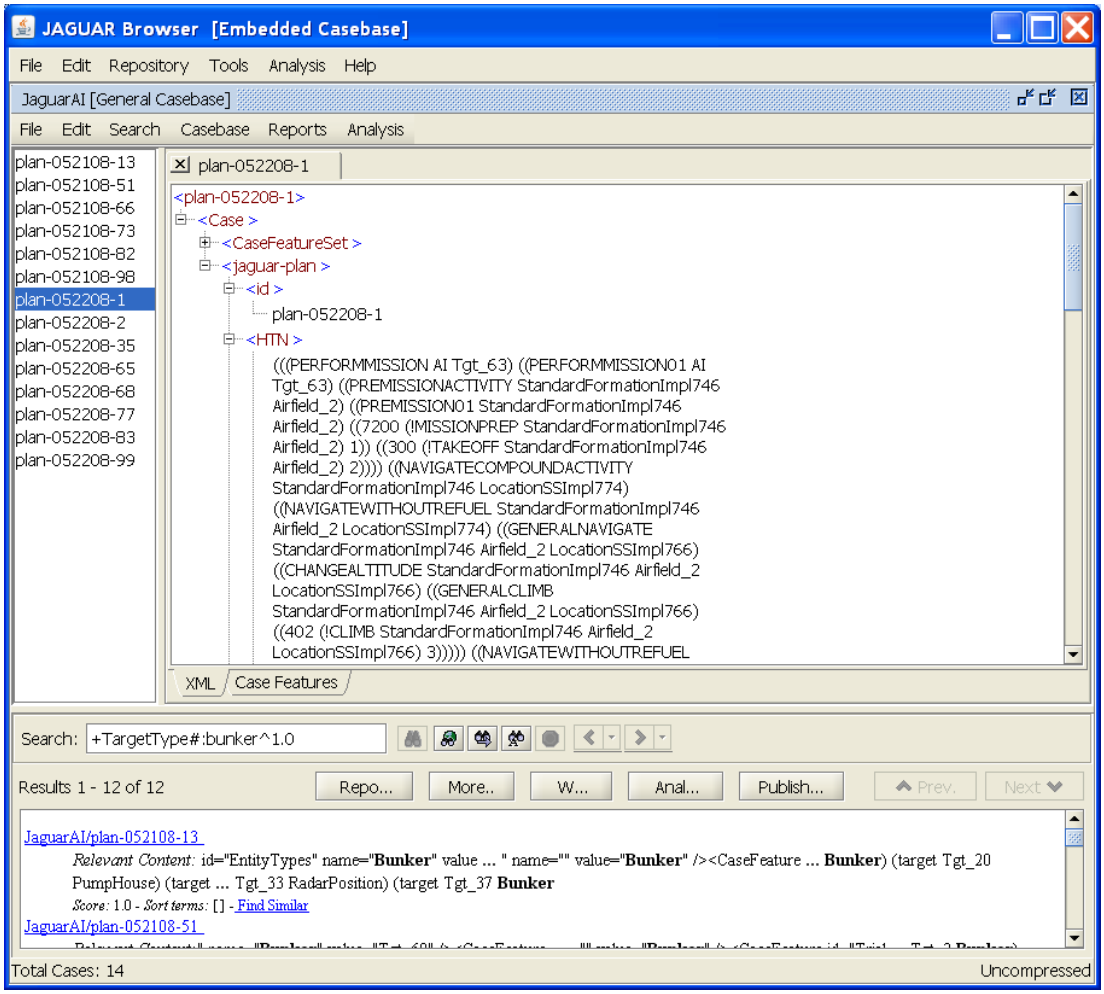

Figure 28. Historical JAGUAR plans are stored in a case base for MICCA.

To support temporal reasoning we created temporal constraint models that were based on the existing JAGUAR process models and their constraints. To realize this, we did the following:

- Represented each action with ***start*** and ***end*** events: (start T1), (end T4) etc.

- Defined optional **temporal milestones** for complex tasks, e.g., (FlightWindowEnd T0). Such temporal milestones enable complex temporal objectives.
- For each domain a set of parametric temporal constraints associated with HTN methods were defined. For each method temporal relationships between subtasks were defined, e.g., Navigation starts right after MissionPrep ends, and temporal constraints were defined, e.g., MissionPrep takes 2hours and the duration of TakeOff depends on the aircraft type.

4.2.1 Agent Configurations

We have utilized all agents described in the Methods section of this paper for this domain. Specific configuration upgrades from the Rovers domain include the following:

- New adaptation agent: Temporal Adaptation Agent.
- New evaluation agents: Risk, Completeness, Executability and any number of custom agents that can be defined by the users.
- Improvements to other agents mostly to support more complex constraints: Instantiation, Adaptability and Plan Adaptation, Merge agent to support aggregation of case features.
- Improvements to coordination agent for supporting complex policies with multiple agents

The LPM used in the Comparison agent for the retrieved plans:

- In decreasing order of importance: Adaptability, Risk, Cost, Completeness
- Higher adaptability and completeness scores are preferred.
- Lower cost and risk scores are preferred.

The LPM used in the Comparison agent for the revised plans:

- In decreasing order of importance: Completeness, Risk, Cost, Adaptability
- Higher adaptability and completeness scores are preferred.
- Lower cost and risk scores are preferred.

4.2.2 Test cases and performance results

We created several test cases with varying levels of difficulty in terms of number and types of objects and also number of conflicts in the problem definition. The following is a summary of our test cases:

- Single objective plans (3 mission types: AI, REC and JSTARS)
- Multiple objective plans
  - 3m simple – 3 objectives, no conflicts
  - 3m – 3 objectives, 3 mission types
  - 4m – 5 objectives, 3 mission types
  - 9m – 9 objectives, 3 mission types
  - 50m – 50 objectives, 3 mission types

Table 4 contains the MICCA performance results for the JAGUAR domain for the multiple objective test cases. Note that this evaluation is not comprehensive. The goal of the evaluation (except for the 50m scenario) was to show that MICCA can produce results under different settings and scenarios within a reasonable amount of time. All of these tests were run on a single laptop (Windows 7, i7 2.2 GHz processor, 3 GB RAM available to MICCA). Since MICCA agents can be run on different machines and parts of the problem can be worked on in parallel, performance will improve if more machines are available. We did not perform any multi machine scalability experiments in this effort. The 50m scenario was designed as a stress test to see if the system could operate with a more complex and big problem. For each test case, MICCA was run once in the mixed-initiative mode and once in the auto-mode. In the mixed-initiative mode the user decides which plans to adapt and merge to form the final COA. In the auto-mode the Merger agents automatically pick candidates to form the final COAs. In general these evaluations indicate the following:

- Although the auto mode is bound by the configuration parameter that controls the number of COAs produced, our results indicate that the mixed-initiative mode tends to generate a smaller number of final revised COAs.

Table 4. JAGUAR Performance Results

Scenario Name	Interaction Mode	*Eval Agents*	Candidate Plans	Objectives	Published	Conflicts	# COAs generated	Uncovered Objectives	Time to Run
3m simple	MI	*4*	7	3	6	0	4	0	2 minutes **
3m simple	Auto	*4*	7	3	7	0	4	0	2 minutes
3m	MI	*4*	7	3	4	3	2	0	2 minutes
3m	Auto	*4*	7	3	7	3	4	0	2 minutes
4m	MI	*4*	10	5	7	4	4	0	2 minutes
4m	Auto	*5*	10	5	10	4	4	0	2 minutes
9m	MI	*4*	25	9	11	2	2	2[1]	10 minutes
9m	Auto	*4*	25	9	23	5	4	2 -3[2]	10 minutes
50	Auto	*4*	25	50	21	100	1*	21[3]	97 minutes

Statistics in this table reflect the usage of MICCA at the end of the project and for each of the scenarios. Auto means that all of the candidate plans were published. MI means that the user interacted and published only the candidate plans of interest.

*One COA was generated because MICCA encounter insufficient resources. The algorithm will only produce one COA if there is no way to produce a complete plan.

**These times vary by as much as ½ the time depending on the computer that is used. The values here represent the runs on a slower machine. The exception is the 50 run, which was run on a fast machine.

1 : (insufficient REC a/c)

2 : (insufficient REC a/c)

3: 21 RECs (there are only 3 REC a/c available); 1 JSTARs (only one available)

All 24 AI objectives were satisfied)

- On average, the agents completed the evaluation and adaptation cycles in no more than 10 minutes.
- The system did complete both cycles for the stress test scenario – 50m. However the running time was 97 minutes. We have identified that most of the time was spent by the temporal adaptation agent due to the size of the problem. We have identified several optimizations (See section 3.4.2) that could improve the performance of this agent in subsequent research.

A number of custom designed reports have been created for the JAGUAR domain. These reports are intended to help the user understand the differences in the COAs that are produced and to view certain aspects of each COA, such as COA validity values that are specified by Air Force doctrine. The following are examples of the current domain specific reports that MICCA can generate.

Feasibility Test (Figure 29): Reports the resources that are assigned to each action in a COA and the status of each resource. To identify the resources MICCA parses the explanations that are part of the revised plans, specifically the explanations generated by the Instantiation agents. To identify the resource status MICCA searches the explanations from the Plan Adaptation agent to see if there are any unverified assumptions regarding the availability of any of the resources. Instead of generating a numeric probability value associated with the availability of the resources at execution, the number of unverified assumptions is provided to the user.

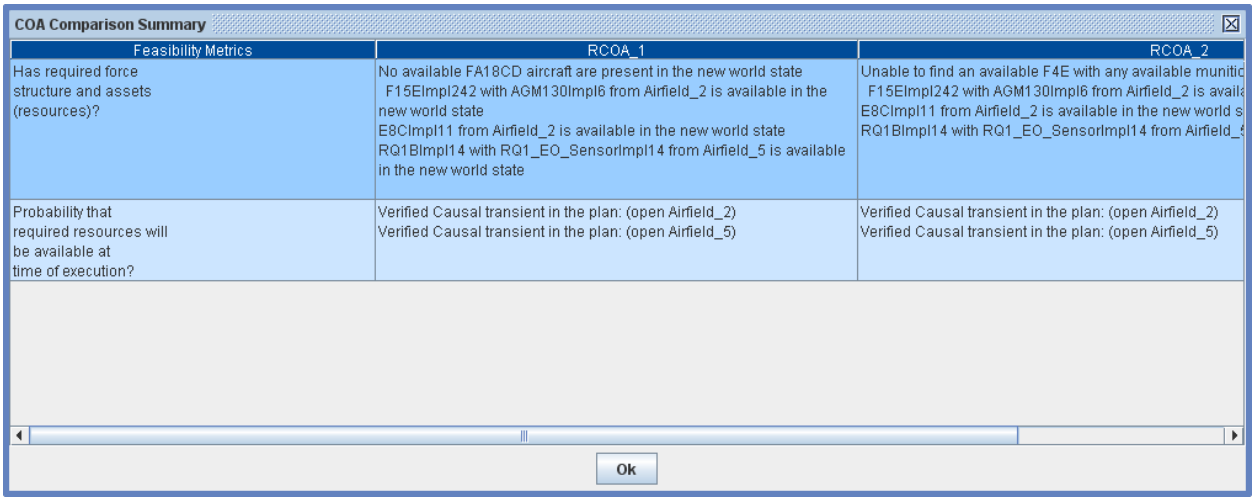

Figure 29. Feasibility report displaying the resources used and their availability for each COA.

Validity Test (Figure 30): The answers to six doctrine based COA questions "who, what, where, when, how, why" are displayed in the Validity report for each COA. This report is generated by parsing the hierarchical plan and the objective statement.

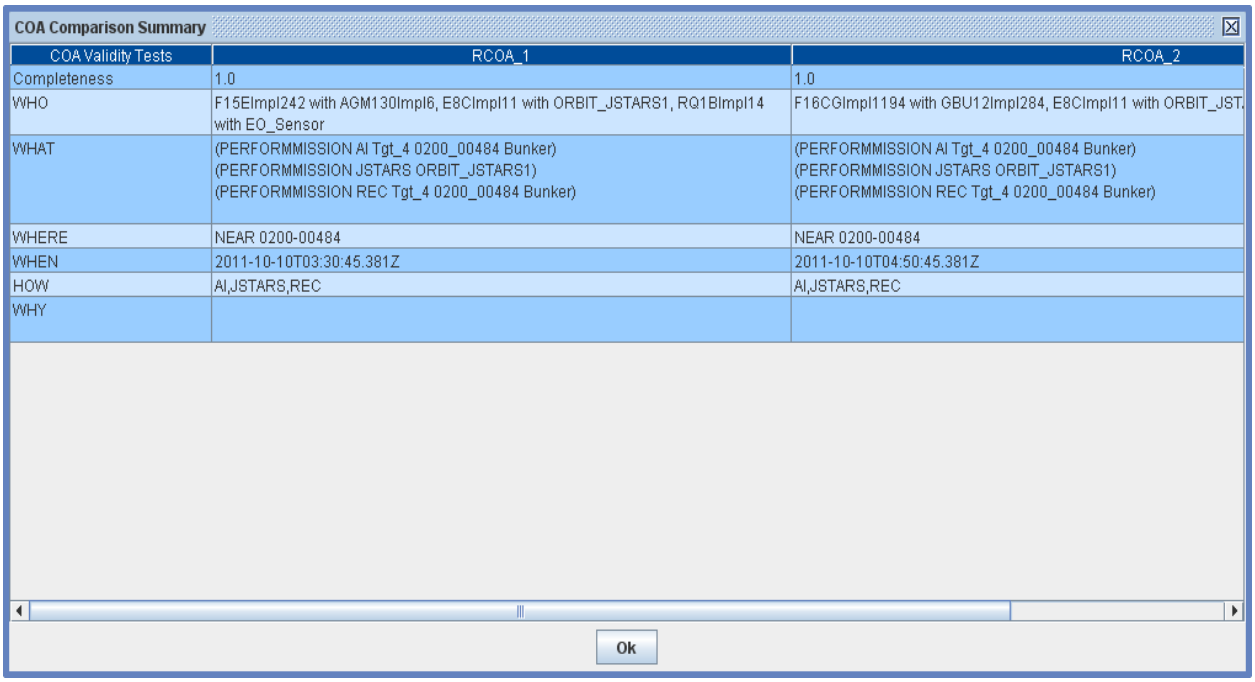

Figure 30. Validity report answering six doctrine based questions.

Finally we conducted a what-if analysis for some of the test-scenarios such as 9m where the resources were not sufficient to accomplish all nine objectives. To simulate a real case scenario where the operator could determine if additional resources could provide a better solution, we allow the user to create a new world state in which there is one more aircraft with the proper capability. If the analysis shows that all of the mission objectives can be completed successfully with the additional resource, then the user could use this information as the basis for requesting more resources to support the actual mission planning.

In the process of these experiments we have realized that the world state contains interdependent information which complicates the creation of a valid scenario by the user. To eliminate potential user errors, and reduce time to complete this task we designed macros and a user interface (Figure 31) that enables the users to perform common actions in one step. Some examples are:

- Make all aircraft of a type X with munitions Y unavailable.
- Close airbase Z.
- Make a new aircraft available with type X, munitions Y, at airbase Z.

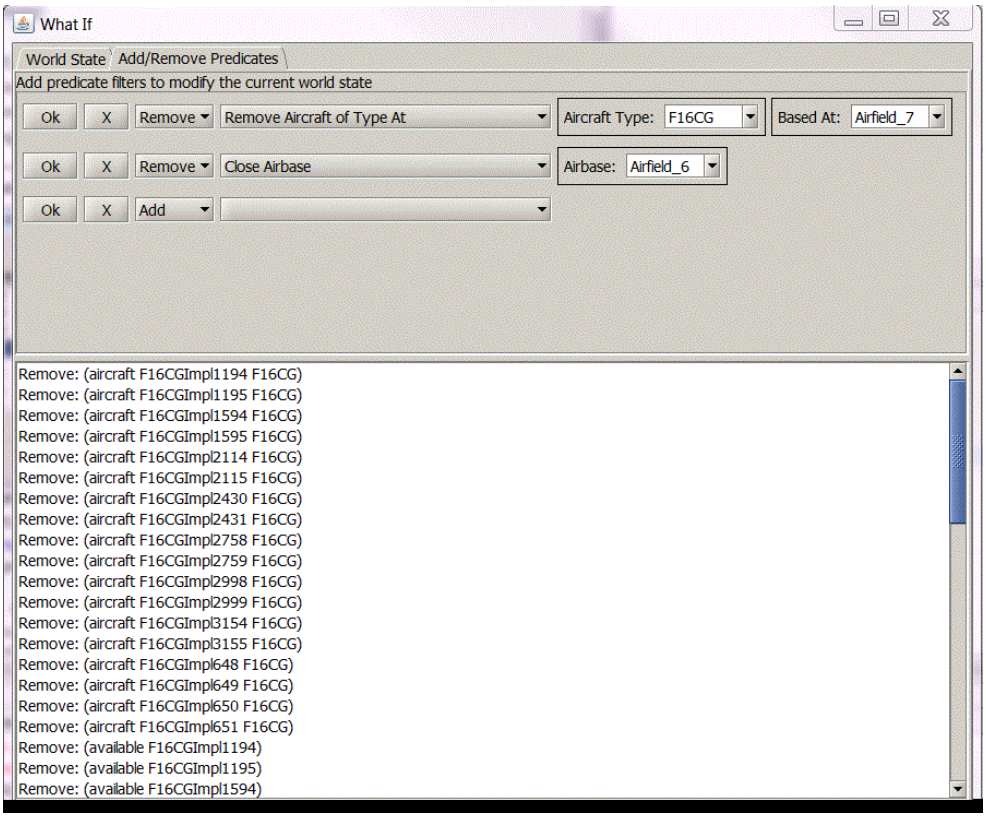

Figure 31. What-if GUI displaying some macros that can help the user more easily create valid hypothetical scenarios.

5.0 CONCLUSIONS

This final report describes the Mixed Initiative Course of Action Critic Advisor (MICCA) research effort. Our MICCA research had two objectives:

- Improve the quality of a course of action that is originally derived from a historical case by realigning it with the current state, objectives and other relevant new information.
- Evaluate a course of action along many dimensions in order to enable the user to make more informed decisions before selecting one for continued planning and execution or to justify any requests for additional resources that could improve the success of a COA in an anticipated future world state.

To achieve these goals we designed and implemented a domain independent agent framework that enables a multi-agent based distributed solution for complex tasks such as adaptation and evaluation. The following agent types have been developed during this project. The function of most of the agents in the following list has been described in this report. Some of the agent types that have not been discussed in detail, such as the Resource Scheduling agents are actually place holders for advanced capability.

- Evaluation Agents
  - Cost Evaluation Agent
  - Risk Evaluation Agent
  - Adaptability Agent
  - Executability Agent
- Instantiation Agents
  - Editing Agent (Resource/capability management)
  - Merger Agent
- Adaptation Agents
  - Planning Agent
  - Temporal Scheduling Agent
  - Causal Agent
  - Resource Scheduling Agent
- Meta-Critic Agents
  - Execution Agent (Evaluation Agent Interaction)
    - Comparison Agent
    - Ranking Agent
  - Adaptation Interaction Meta-Critic
    - Coordination Agent
- Enabling Agents
  - Problem Definition Agent
  - Case Base Publishing Agents

- Browser Publisher Agent
- Candidate Case Retriever Agent

A major contribution of this work is the design of the meta-critic framework, which defines the communication protocols that are required in order to ensure the reliable, correct and complete execution of the tasks. The mixed initiative aspects of MICCA utilize the human as another agent for problem solving. The proof of concept MICCA implementation utilizes the DEEP publication/subscription framework for agent communication.

We believe that the current MICCA capabilities can be leveraged to support other levels of planning, e.g., operational level COA evaluation and adaptation. However, our results indicate that MICCA should be improved along the following dimensions:

- Optimizations of the framework and agents. The most immediate need for optimization as shown by the evaluations is the temporal reasoning.
- Improved user interfaces: The user interfaces in MICCA are basic and focus on functionality rather than usability. With the help of subject matter experts and user studies we can improve the mixed initiative aspect of MICCA considerably.
- Robustness of the system: The proof of concept implementation can be further tested and improved using systematic tests and redesigned to for use as a service.

6.0 REFERENCES

[1] Carozzoni, J. A., Lawton, J. H., DeStefano, C., Ford, A. J., Hudack, J. W., Lachevet, K. K., and Staskevich, G. R., Distributed Episodic Exploratory Planning (DEEP), AFRL Technical Report, AFRL-RI-RS-TR-2008-279, October 2008.

[2] Mulvehill, A. M., Benyo, B., Cox, M., and Bostwick, R., Expectation Failure as a Basis for Agent-Based Model Diagnosis and Mixed Initiative Model Adaptation during Anomalous Plan Execution, Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, January 6 - 12, 2007, Hyderabad, India, AAAI Press, Menlo Park, CA, 2007; pp. 489-494.

[3] Nau, D. S., Au, T. C., Ilghami, O., Kuter, U., Muñoz-Avila, H., J. W. Murdock, J. W., Wu, D., and Yaman, F., Applications of SHOP and SHOP2. IEEE Intelligent Systems 20(2): pp. 34–41, Mar.-Apr. 2005.

[4] Veloso, M. M, Mulvehill, A., Cox, M., "Rationale-Supported Mixed-Initiative Case-Based Planning", IAAI Conference Proceedings, August 1997.

[5] Muscettola N., Nayak P.P., Pell B., and Williams B.C., Remote agent: to boldly go where no AI system has gone before. Artificial Intelligence, 103(1-2):5–48, August 1998.

[6] Berry P.M., Gervasio M., Uribe T. E., Pollack M. E., and Moffitt M. D., A Personalized Time Management Assistant. AAAI Spring Symposium, March 2005.

[7] Pollack M. E., and Tsamardinos I., Efficiently Dispatching Plans Encoded as Simple Temporal Problems. In Intelligent Techniques for Planning, 2005.

[8] Dechter R. , Meiri I., Pearl J., Temporal constraint networks, Artificial Intelligence, v.49 n.1-3, p.61-95, May 1991.

[9] Allen J. F., Maintaining knowledge about temporal intervals. *Communications in. ACM*, 26(11):832–843, Nov. 1983.

[10] Fishburn, P., 1974. Lexicographic orders, utilities and decision rules: A survey. Management Science 20 (11), 1442–1471.

[11] Gigerenzer, G., Goldstein, D. G., 1996. Reasoning the fast and frugal way: Models of bounded rationality. Psychological Review, Vol. 103.

[12] Yaman, F., desJardins, M., More-or-Less CP-Networks. Uncertainty in Artificial Intelligence, Vancouver, Canada, Jul 20-22, 2007.

[13] Aamodt, A. and Plaza, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Com – Artificial Intelligence Communications, 7 (1): 39-59, March 1994.

[14] Ayan, N. F, Kuter, U., Yaman, F. and Goldman, R. HOTRiDE: Hierarchical Ordered Task Replanning in Dynamic Environments, ICAPS 2007 Workshop on Planning and Execution for Real-World Systems, Providence, RI, Sep 22, 2007.

[15] Floyd, R. W. "Algorithm 97: Shortest Path". Communications of the ACM 5 (6): 345, June 1962.

[16] Long D. and Fox M. "The 3rd International Planning Competition: Results and Analysis", Artificial Intelligence Journal (AIJ), Volume 20, pages 1-59, 2003

APPENDIX-A: FILE AND CASE BASE DESIGN

In the DEEP system case bases contain historical plan information that can be used by plan evaluation and plan adaptation agents to support COA development. In order to facilitate the development of the MICCA agents in the spirit of DEEP, we have designed a process that specifies how human operators and/or software agents would make use of the case base data in search of potential candidate plans. In order for the historical data to be retrieved, the case plan data must have certain attribute-value pairs to allow a human operator or software agent to (1) specify and execute the query and (2) perform some similarity matching between the retrieved cases and the current state and objectives. Furthermore, the historical plan data within the case base must contain some attribute value pair that can be used to distinguish one case from another, e.g., PlanID.

For any given domain, plan data in a case base can contain the plan, part of a plan with a pointer to the entire plan, the plan with the world state, or any other combination. For example, in JAGUAR, the plan is a continuous entity that is comprised of one or more missions. Hence the plan case base for JAGUAR is a case base of missions with information on how they were executed during a given "trial". A trial can be viewed as a plan with a given time interval, e.g., all of the missions that were executed for a given day. The initial plan data and the world state are stored in external message archives that need to be loaded if necessary. In the Rovers domain (the simple domain we are using to test the development of MICCA agents), two case bases were developed to support development. The "Rover Task" case base contains historical plan data that describes how Rovers were used to accomplish certain goals. The case base attributes for the Rover Task case base include:

Full goal	;;; full goal with objective
GoalType	;;; type of task
Objective	;;; entity instance associated with task
PlanID	;;; string that describes the name of the plan
Rover	;;; agent that does the task
RoverOrigin	;;; waypoint the Rover started at
PlanLength	;;; Total number of actions in the plan
VisitedWaypoint	;;; Waypoint visited by the rover during plan execution

The PlanID in the Rover Task case base is a unique ID that identifies the plan that the rover participated in.

The second case base for the Rover domain contains the plan data associated with the plan. This includes the following:

- PlanID
- Plan
- World State
- HTN

APPENDIX-B: KNOWLEDGE BASES AND OTHER FILE BASED INPUT FORMAT

TABLE OF CONTENTS

SECTION **PAGE**

There are several common inputs to every evaluation and adaptation agent. In general the agents don't interact with raw input, i.e., a file containing a string in a specific format. Instead they utilize data that is refined by several MICCA parsers that can process strings that represent an objective, world state or plan. This section explains the expected format for these inputs.

B.1 World State

The *World state* is a list of ground (i.e., without any variables) predicates in lisp format, i.e., prefix notation with enclosing parenthesis. The following gray box provides an example World State that describes the availability of FA18CD aircraft:

```
((inFormation SFI5774 FA18CDImpl1901)
(available FA18CDImpl1901)
(in FA18CDImpl1900 Unit_2)
(in FA18CDImpl1901 Unit_2)
(aircraft FA18CDImpl1900 FA18CD)
(aircraft FA18CDImpl1901 FA18CD))
```

B.2 Knowledge Base

The *knowledge base* is used in MICCA to contain data that is similar to the data contained in the world state and thus in the implementation, both are represented by the same data structure. The main difference between a knowledge base and a world state is that the facts about the world that are stored in the knowledge base do not tend to change quickly; such as the maximum speed of an aircraft. The following gray box provides an example of a Knowledge Base that describes some of the characteristics of an FA18CD:

```
((speed FA18CD GeneralStrike 450 550)
(speed FA18CD Navigate 450 475)
(takeOffTime FA18CD 300)
```

Information about the World State that is likely to change due to external events is stored in a transient knowledge base. For example the operating hours of a base could change as a function of an approaching weather event.

B.3 Plan

MICCA plans are hierarchical which means that they are a list of decomposition trees. In addition to the hierarchical plan, the plan has a flat plan component, which is basically a list of all the leaves of the decomposition trees. While the flat plan can be generated from the hierarchical plan, in MICCA we pre-compute and store the flat plan separately to save computation time. This is simply because some of the agents operate on flat plans only and re-computing the flat plan in every cycle would waste time and result in decreased performance. An example of a Hierarchical Plan is provided in the following gray box in which the top level task is the PreMissionActivity which is accomplished by the task PreMission01 which consists of two

simple tasks: MissionPrep and TakeOff. The numbers preceding the simple actions are the costs associated by the action. The numbers following the action names are execution order, start and end time windows.

```
((PREMISSIONACTIVITY StandardFormationImpl5774 Airfield_1)
  ((PREMISSION01 StandardFormationImpl5774 Airfield_1)
       (7200 (!MISSIONPREP StandardFormationImpl5774 Airfield_1)  151
                          2005-05-09T12:23:42.381Z 2005-05-09T14:23:42.381Z)
       (300 (!TAKEOFF StandardFormationImpl5774 Airfield_1)  152
                         2005-05-09T14:23:42.381Z  2005-05-09T14:28:42.381Z)))
```

An example of a Flat Plan is provided in the following gray box which contains only the simple actions.

```
 ( (7200 (!MISSIONPREP StandardFormationImpl5774 Airfield_1)  151
                              2005-05-09T12:23:42.381Z 2005-05-09T14:23:42.381Z)
   (300 (!TAKEOFF StandardFormationImpl5774 Airfield_1)  152
                              2005-05-09T14:23:42.381Z  2005-05-09T14:28:42.381Z))
```

B.4 HTN Models (Methods, Operators)

We have adapted the HTN domain format that is used in the SHOP planning algorithm. Specifically we have adapted the JSHOP dialect of that language because we use the java implementation of the SHOP algorithm which slightly deviates from the lisp convention. In this section we provide examples of how the components of a planning domain, e.g., methods, operators, and axioms are implemented in MICCA.

Example Method:

```
(:method (PreMission01 ?actor ?airfacility)
  ()
  ((!MissionPrep ?actor ?airfacility)
   (!Takeoff  ?actor ?airfacility)))
```

Example Operator:

```
(:operator  (!MissionPrep  ?flightEntity  ?airFacility)
   ((inFlight  ?flightEntity false) (available ?flightEntity) )
    ((available ?flightEntity) )
    ((prepcomplete ?flightEntity)))
```

Example Axiom:

```
(:- (inBattleSpace ?point)
    ;; case 1: point is a target
     ((target ?point ?type))
    ;; case 2: point is a strike point for StandIn strike
     ((Strikepoint ?point ?target))
    ;; case 3: point is first of the standoff strike points
     ((standOffStrikePoints ?point ?s ?target))
    ;; case 4: point is second of the standoff strike points
     ((standOffStrikePoints ?s ?point ?target)))
```

B.5 Temporal Constraints

Temporal constraints in MICCA are quantitative. Temporal constraints for methods and operators are defined to impose either temporal ordering on the subtasks of a method or durations on the operators. Following is an example of a Temporal Constraint for Methods:

```
(:method (PreMission01 ?actor ?airfacility) t ()
 (( (!MissionPrep ?actor ?airfacility) (!Takeoff  ?actor ?airfacility))
 (t0 t1)
 ((eq (start t) (start t0))
   (eq (start t1) (end t0))
   (eq (end t) (end t1)))))
```

The following is an example Temporal Constraint for Operators:

```
(:operator  (!Takeoff  ?flightEntity ?airFacility) a
   ((inFormation ?flightEntity ?plane)
     (aircraft ?plane ?acType)(takeOffTime ?acType ?t))
   ((eq (dur a) ?t))
)
```

B.6 Commander's Intent

This is a text file in natural language. The contents of this file can be viewed by the human operator in order to better understand the overall problem, and this data is used by the MICCA operator to manually convert the content into the "objectives file".

B.7 Objectives

Objectives are task lists with associated temporal constraints. The objective may contain other parameters that are of relevance to the domain, e.g., the target instance ID, mission type, a priority value, etc. The following is an example Objective:

```
( ((T1 1 (PERFORMMISSION JSTARS ORBIT_JSTARS1)))
((geq (ObjectiveWindowStart T1) "2011-10-10T05:10:00Z")
 (leq (ObjectiveWindowEnd T1)   "2011-10-10T11:45:00Z")))
```

In this example the first element (*T1*) is the label of the task, the second element (*1*) is the number indicating the priority – one being the most important, the third is the task and finally the temporal constraints referring to the milestones of a task using its label.

B.8 Preference Models

The Comparison agent utilizes preference models when comparing two plans based on their scores. These preference models can be modified by the user however MICCA expects two preference models as input: one for comparing the retrieved/candidate plans and the other one for comparing the revised plans. MICCA currently supports reasoning with lexicographic preference models with numeric attributes only. .An example LPM defined for JAGUAR domain is shown below.

```
(JaguarLPM
 (MICCAAdaptibility number more 1)
 (MICCACost number less 3)
 (MICCACompleteness number more 4)
 (MICCARisk number less 2))
```

B.9 Cases

Historical plans are stored in an XML data format that consists of five pieces of information: a unique string plan ID, the HTN plan (in the format described in Section 3.8.3), a set of case features, the world state that the plan was created for, (in the format described in Section 3.8.1), and the set of objectives the plan was created to satisfy (Section 3.8.6). The case features are set of feature names and values that represent the meta data about a plan and can be used to facilitate searching. The following grey box provides an example of some of the case features of a JAGUAR plan.

```
<?xml version="1.0" encoding="UTF-8"?>
<jaguar-plan>
  <id>plan-052308-0427-107</id>
  <HTN>(PERFORMMISSION AI Tgt_61 "0200_00266" Bunker) … </HTN>
  <CaseFeatureSet><CaseFeature id=”MissionType” value=”AI”/>…
  <initial-state>(DEFPROBLEM plan-052308-0427-107 JAGUAR ((
                    (target Tgt_1 Bridge) …
                   (ORDERED (T1 1 (PERFORMMISSION AI Tgt_61 …
  </jaguar-plan>
</xml>
```

APPENDIX-C: INPUT/OUTPUT MESSAGE FORMATS

TABLE OF CONTENTS

SECTION **PAGE**

C.1 Candidate Plan

Candidate plans are retrieved by the Candidate Case Retriever agent and published on the blackboard at the beginning of a scenario. Then the *Instantiation Agents* and *Coordination Agent* process candidates in order. The instantiation agents are responsible for producing plans that are guaranteed to be compatible with the current objective. The candidate plans have the following format:

<Plan, SourceID, ObjectiveID, StateID, Status>

- **Plan**: The plan that is to be revised by the adaptation agents.
- **SourceID:** The unique identifier for the source of this plan. The source will reveal information about the domain of this plan.
- **ObjectiveID:** ID that is used to obtain the objectives.
- **StateID:** ID that contains the new world state information.
- **Status:** Represents the state of the candidate: Retrieved, Edit, Merge, Adaptation. Depending on the state of the plan the next agent that will process the plan is determined.

A retrieved plan is simply a candidate that has the status: “Retrieved”.

C.2 Revised Plan

Once the coordination agent decides that the revision cycle is over, it will turn the current candidate plan into a revised plan. The revised plan has the same structure as the Candidate Plan with extra fields such as revision summary and adaptation statistics.

C.3 Scored Plan

The Evaluation Agents evaluate the plans (retrieved or revised) with respect to some criteria and output a Scored Plan in the following format:

<**PlanID, SourceID, ObjectiveID, StateID, AgentID, Score, Confidence, Justification**>

- **PlanID**: The unique identifier of the evaluated plan (revised or retrieved).
- **SourceID, ObjectiveID, StateID**: same as input.
- **AgentID**: The unique identifier of the agent that produced the score.
- **Score**: Number representing the evaluation score. The direction of good/bad is unknown.
- **Confidence**: Double between 0 and 1; where 0 represents a poor confidence score and 1 represents the highest confidence.
- **Justification**: String stating the reason for the confidence. This will be generated from predefined templates. The contents of the justification are specific to the agent. For example for the Adaptability agent, this may include reference to the total preconditions, including the preconditions that have been satisfied. If no preconditions have been satisfied, the value is “none”.

C.4 Plan Pair Comparison

The output of the Comparison Agent has the following format:

<PlanID1, PlanID2, SourceID, ObjectiveID, StateID, Result, Justification>

- PlanID1 and PlanID2: Identifiers of the plans that are compared.
- SourceID, ObjectiveID, StateID: come from the Scored Plans that give the scores for these two plans in the same context.
- Result: 0 if the plans are equally good. 1 if the first plan is better, -1 otherwise.
- Justification: The rationale for the result.

C.5 Ranked Plans

This is an ordered list of plans with the following structure:

<PlanID, SourceID, ObjectiveID, StateID, Ranking>

- PlanID: Identifier for the selected plan.
- SourceID, ObjectiveID, StateID: come from the input PlanPairComparison objects.
- Ranking is the ranking of the plan – lowest being the highest ranked.

C.6 Selected Candidate or Revised Plans

If the plans are retrieved (identifiable by the Status) then this will trigger the adaptation cycle beginning with the Instantiation agents. Otherwise the output will be an execution candidate, which will be published to the DEEP Blackboard where it is then available to other systems, e.g., the case base, a simulation system, or an external plan execution system.

C.7 Revision Request

Upon receiving a candidate plan, the coordination agent will issue a Revision Request for particular adaptation agents. The reasoning behind how/when to assign an agent to a specific task is dictated by the coordination policy preferred by the user and the adaptation tasks required by the domain. Revision Requests have the following format:

< **CandidatePlanID, RevisionNumber, AgentID, RequestType**>

- **CandidatePlanID**: The ID of the candidate plan that is being revised. Essentially this acts as the task identifier.
- **RevisonNumber**: This number is used for version control of the revisions which will be necessary if one of the agents hits a dead end and backtracking is required.
- **AgentID**: The agent that this revision request will be processed by.
- **RequestType**: This is a complex data structure by itself. It can be either a new request for a plan or a revision request for a previous revision response. In general the structure will be a tuple: **<TaskType, Data>**
  - **TaskType**: *FirstRevision* or *Rerevision.* If the agent is asked to produce a different solution for a previous request then the TaskType is Rerevision. Otherwise the type is FirstRevision.

- **Data**: For the FirstRevision the data is the plan, for Rerevision the data is simply a ResponseID.

C.8 Revision Response

The adaptation agent will respond to a Revision Request with a Revision Response. Depending on the request type, the agent will either operate on new conflicts that were already in the candidate plan (or introduced during other revision cycles) or produce a different solution that will take into account new constraints. The Revision Response has the following format:

< **CandidatePlanID, RevisionNumber, AgentID, Response**>

- **CandidatePlanID**: The ID of the candidate plan that is being revised. Essentially this acts as the task identifier.
- **RevisonNumber**: This number is used for coupling this response to the request.
- **AgentID**: The agent that produced this request.
- **ResponseType**: This is a complex data structure because it represents four kinds of response an agent can produce:
  - **Failure:** The plan contains a conflict that cannot be resolved. Also reports the unresolved conflicts and partial solutions if possible.
  - **Success:** The plan contains no conflicts and all issues are resolved after this revision. The data that is returned is a revised version of the plan that the agent received in the Revision Request.
  - **Partial:** The plan still requires revisions but additional information is needed. This is different than conflicts. For example a plan that contains high level actions still needs to be revised so that the details of the plan can be finalized.

APPENDIX-D: XML DATA EXTRACTION

For the JAGUAR domain, the data available (historical plans, world state, objectives) is present in a set of XML files that were extracted from JAGUAR trial runs. In order to use this data in MICCA, relevant information needed to be extracted from the XML tree and converted to the predicate style syntax. We developed a general extraction method that could easily be extended to extract additional data, or could be used for an entirely different domain that has data stored in XML.

The extraction procedure is implemented as a set of XPATH transformation objects. Each transformation object consists of the following:

- Main XPATH query that selects a subset of the XML tree.
- Predicate pattern with variables.
- Variable definitions mapping variables in the predicate pattern with further XPATH queries.
- Additional flags to control the transformation.

As an example, the transformation for the Takeoff activity in the JAGUAR domain is given below.

```
# Takeoff
xform.4.file=plan
xform.4.pattern=(!TAKEOFF ?flightEntity ?airFacility ("?start" "?end") ?duration ?id)
xform.4.objectXpath=/:PlanRealized/:PlanSet/:PlanBranchView/:ActivitySet/:Activity[
@type="Takeoff"]
xform.4.var.1=flightEntity
xform.4.xpath.1=jpm:Participants/jpm:EntityRef[@role="flightEntity"]/@id
xform.4.var.2=airFacility
xform.4.xpath.2=jpm:Participants/jpm:EntityRef[@role="airFacility"]/@id
xform.4.var.3=id
xform.4.xpath.3=@id
xform.4.var.4=duration
xform.4.xpath.4=jpm:TimeInterval/jpm:Duration/@nominalDuration
xform.4.var.5=start
xform.4.xpath.5=jpm:TimeInterval/jpm:StartTime/@nominalDTG
xform.4.var.6=end
xform.4.xpath.6=jpm:TimeInterval/jpm:EndTime/@nominalDTG
```

A file parameter defines which type of XML file to execute this transformation on. In the JAGUAR domain, we use the World State, Problem Statement, Plan, and Model knowledge base to extract information from. The pattern property defines the predicate(s) that this transformation will create. Symbols in the pattern prefixed by a "?" indicate variable names. The objectXpath property defines an XPATH query that returns a set of XML objects. One predicate will be generated for each such object using the defined predicate pattern. The var.X

and xpath.X properties define an XPATH query to determine the value of a specific variable in the predicate. This XPATH sub query is executed on the XML object returned by the objectXpath query. If there are multiple results for a variable sub query, the generated predicate is cloned, and one instance is created for each result.

For example, the first argument of the predicate given in the example above is:

?flightEntity.

The value for this variable is determined through executing the XPATH query:

"jpm:Participants/jpm:EntityRef[@role="flightEntity"]/@id",

which gets the id of the "flightEntity" role Entity Reference in the participant list of an XML object. The XML object used is the result of the XPATH query:

"/:PlanRealized/:PlanSet/:PlanBranchView/:ActivitySet/:Activity[@type="Takeoff"]",

which gets all activities of type "Takeoff" in an activity set.

There are a set of additional flags that can be used to control this transformation. A flag can be defined for a specific variable transform, or for the entire predicate transformation.

- **Mustexist**: When defined for a specific variable, if that variables XPATH query produces no results, no predicate is generated. The default behavior is to replace any unbound variable with the empty string, and still generate the predicate.
- **Postprocess**: Provides a java method that will be called after the transform is complete. This java method must take a Vector of Strings as input. It will be passed a Vector of the string from of all the predicates generated so far, by any transformation. This java method can modify these strings to provide additional transformations that can't be expressed in the property language.
- **Datatype**: The default behavior is to create one String for each variable query. If there are multiple values returned by a query, they are concatenated by the vertical bar separator into one string. Instead, setting datatype to "nodeset" for a variable will cause multiple predicates to be generated for multiple values, one predicate for each value.
- **Generator**: Instead of performing an XPATH query to get the value of a variable, this generates the value using the given pattern. The pattern can be any string with the substring ?INDEX included. ?INDEX will be replaced with each time the value is generated with an incrementing integer index.
- **Global**: This variable and value are globally defined. Later transformations can use this variable and the value will be remembered and substituted in without needing to do another query.

The benefit of this design is that it allows additional data to be extracted from a large data source (such as the JAGUAR XML plans) with minimal additional effort. As new agents are implemented that require some new predicates, additional transforms can be quickly generated. Then the process of generating the final MICCA HTN data from the input XML data is fully automated. The *XpathToPredicateConverter* utility takes the set of transforms defined, executes them, and produces MICCA HTN world state and plan files. An additional utility, called the *CasebaseGenerator*, then takes these MICCA plan files and produces the proper input format for the JAGUAR Case base.

APPENDIX-E: ASSORTED GUI FIGURES

LIST OF FIGURES

FIGURE **PAGE**

In this section some additional GUIs are presented that were not provided in the body of the paper.

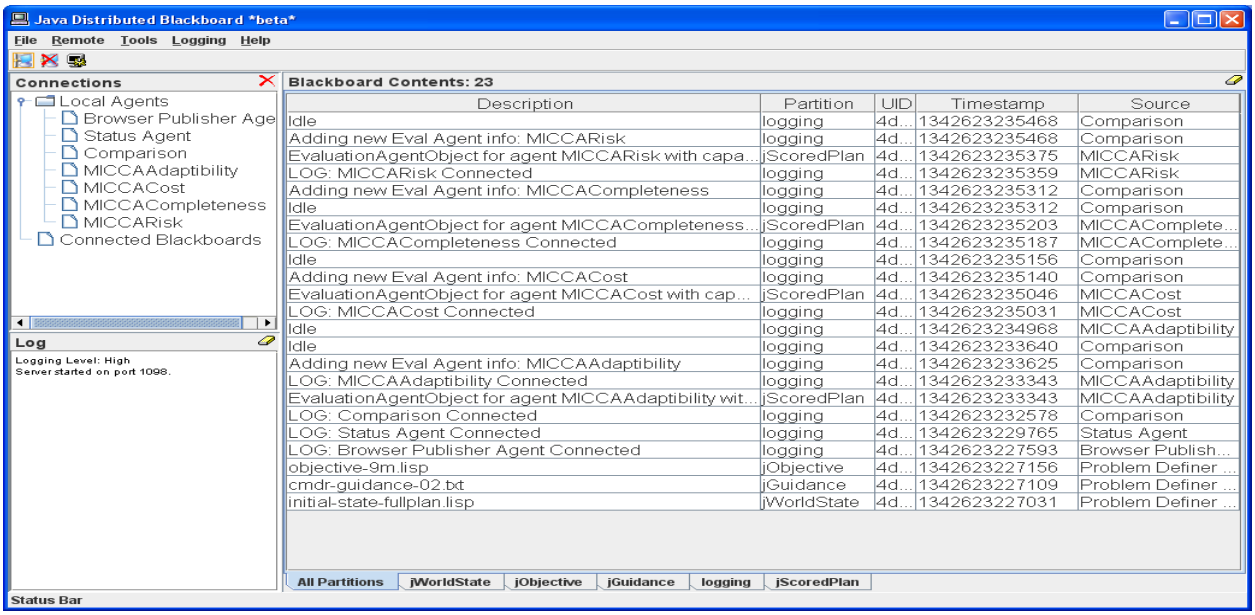

Figure E-1. Blackboard Messages

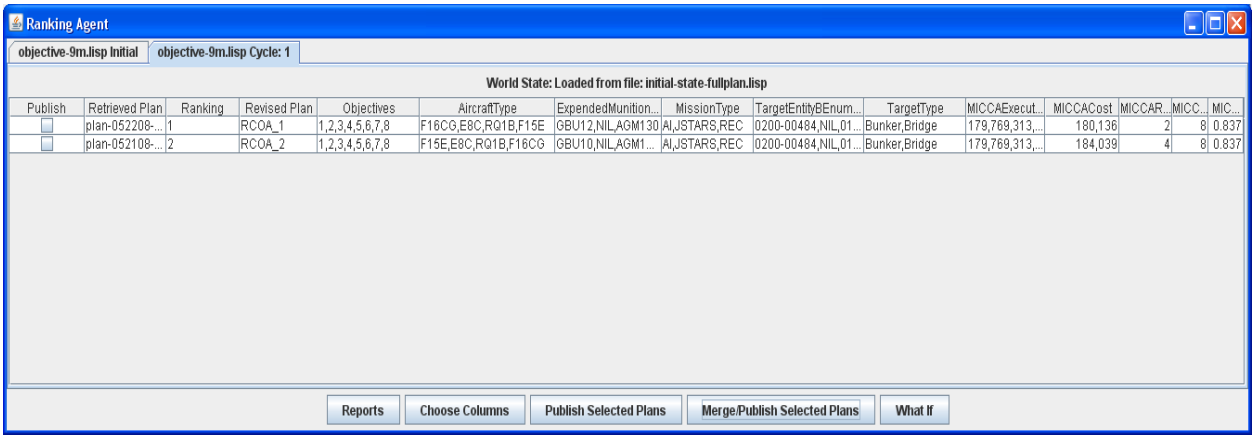

Figure E-2. RCOA Results

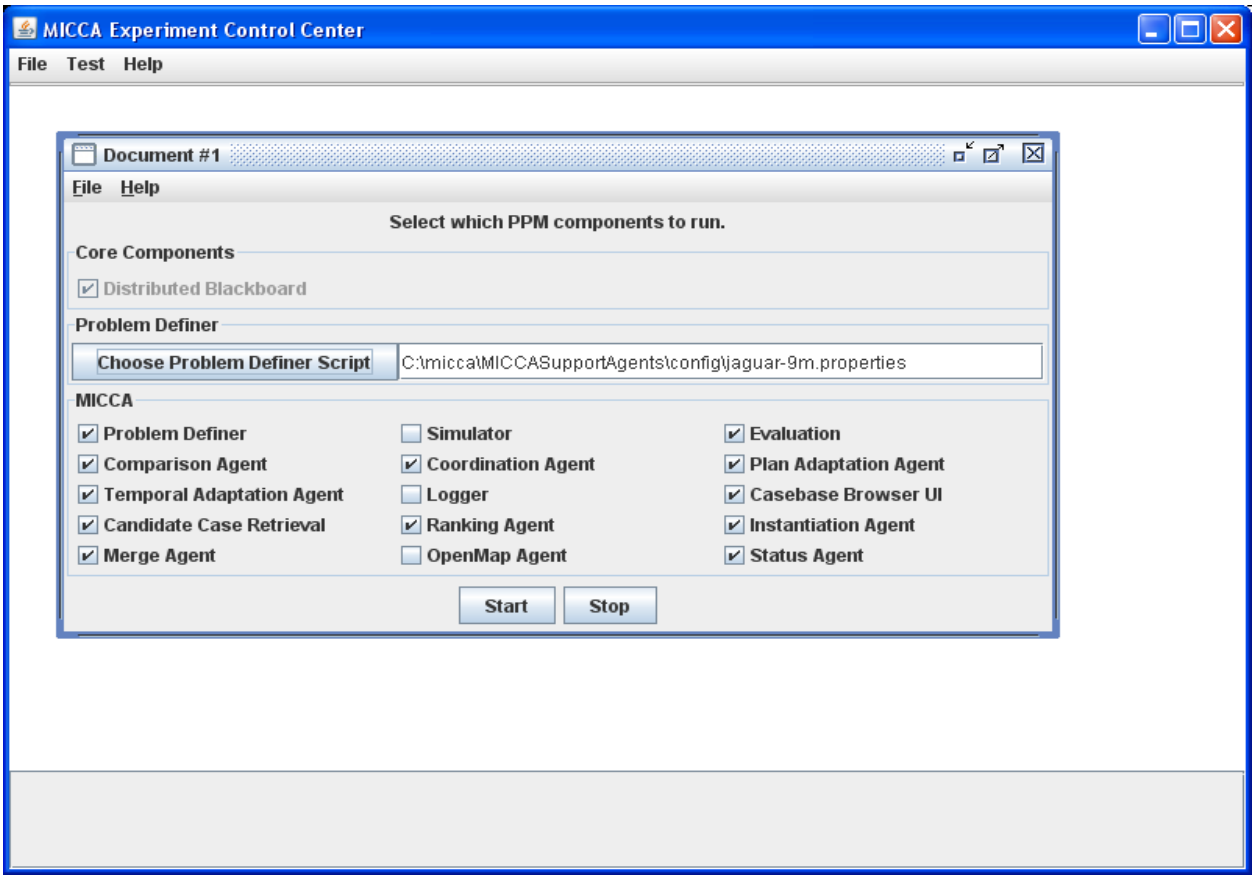

Figure E-3. MICCA Experiment Control Center

LIST OF SYMBOLS, ABBREVIATIOINS, AND ACRONYMS

AFRL .....Air force Research Laboratory

AI .....Air Interdiction

API .....Application Program Interface

ATO .....Air Tasking Order

Auto.....Automated

BB .....Blackboard

BNF.....Backus Naur Form

CBR.....Case Base Reasoning

CI.....Commander's Intent

COA .....course of Action

COTS .....Commercial Off The Shelf

DARPA .....Defense Advanced Research Projects Agency

DEEP.....Distributed Episodic Exploratory Planning

DoD.....Department of Defense

GUI .....Graphical User Interface

HCI.....Human Computer Interface

HotRiDE .....Hierarchical Ordered Task Replanning in Dynamic Environments

HTN .....Hierarchical Task Network

I/O .....Input/Output

ID .....Identification

JAGUAR.....Joint Air/Ground Operations: Unified, Adaptive Replanning

JSHOP.....Java Version of Simple Hierarchical Ordered Planner

JSTARS.....Joint Surveillance Target Attack Radar System

KB .....Knowledge Base

Loc .....Location

LPM .....lexicographic preference model

M .....Mission

MI.....Mixed-Initiative

MICCA .....Mixed-Initiative Course-of-Action Critic Advisors

MissionPrep Mission Preparation

N/A.....Not applicable

NASA.....National Aeronautics and Space Administration

Obj.....objective

PDDL .....Planning Domain Definition Language

QScore.....Quality Score

RCOA .....Revised Course of Action

REC.....Reconnaissance

SExp.....Symbol Expression

SHOP .....Simple Hierarchical Ordered Planner

STN.....Simple Task Network

TGT.....Target

UI .....User Interface

w.r.t. .....with respect to

WS.....World State

XML.....Extensible Markup Language

XPATH .....XML Path Language